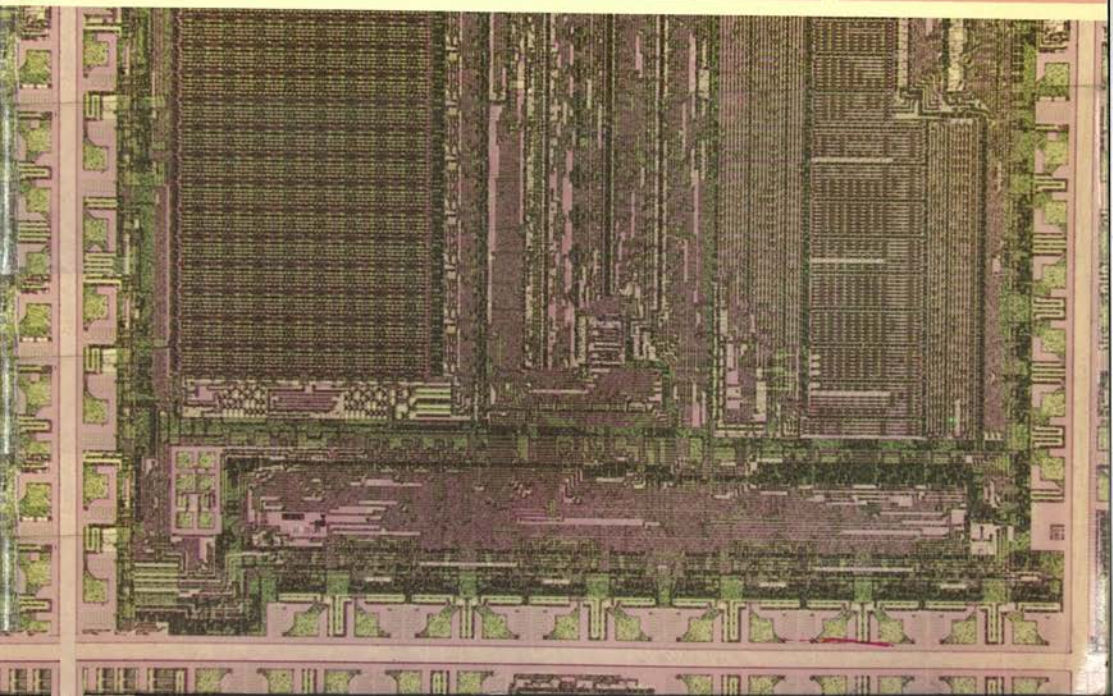
Programmazione dell'8080 e progettazione logica

EDIZIONE ITALIANA

ADAM OSBORNE

GRUPPO EDITORIALE JACKSON



Programmazione dell'8080 e progettazione logica

di Adams
Osborne

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

© Copyright per l'edizione originale Osborne/McGraw-Hill, Inc. 1976

© Copyright per l'edizione italiana Osborne/McGraw-Hill, Inc. 1981

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana le signore Francesca di Fiore, Rosi Bozzolo e l'Ing. Sergio Zannoli. Traduzione a cura di eds electronic data service - Bresso (Mi).

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

SOMMARIO

Capitolo		Pagina
1	INTRODUZIONE	1-1
	CHE COSA SI PRESUME NOTO PER LA LETTURA DI QUESTO LIBRO	1-2
	COMPRENSIONE DEL LINGUAGGIO ASSEMBLY	1-2
	FORMA DELLA STAMPA DI QUESTO LIBRO	1-2
2	LINGUAGGIO ASSEMBLY E LOGICA DIGITALE	2-1
	IL CICLO DI PROGETTO	2-1
	SIMULAZIONE DELLA LOGICA DIGITALE	2-4
	SIMULAZIONE AL MICROCALCOLATORE DI UN INVERTITORE DI SEGNALE	2-5
	UNA SEQUENZA DI EVENTI DI UN MICROCALCOLATORE	2-5
	REALIZZAZIONE DELLA FUNZIONE DI TRASFERIMENTO	2-6
	DETERMINAZIONE DELLE SORGENTI E DESTINAZIONI DEI DATI	2-6
	TIMING DI EVENTO	2-11
	BUFFER, AMPLIFICATORI E CARICATORI DI SEGNALE	2-13
	SIMULAZIONE AL MICROCALCOLATORE DEGLI INVERTITORI ESADECIMALI 7404/05/06/07	2-19
	SIMULAZIONE AL MICROCALCOLATORE DEI 7408/09 DUE PORTE QUADRUPLO POSITIVO GATES AND	2-20
	DUE FUNZIONI D'INGRESSO	2-21
	LA SIMULAZIONE AL MICROCALCOLATORE DI UN 7411 TRIPLO, TRE-INGRESSI, GATE AND POSITIVO	2-22
	TRE FUNZIONI D'INGRESSO	2-22
	MINIMIZZAZIONE DEGLI ACCESSI AI REGISTRI CPU	2-24
	CONFRONTO DELL'IMPIEGO DI MEMORIA E DELLA VELOCITA' DI ESECUZIONE	2-26
	SIMULAZIONE AL MICROCALCOLATORE DI UN FLIP-FLOP 7474 DUALE, TIPO TRIGGER A GRADINO POSITIVO CON PRESET E CLEAR	2-27
	UNA DESCRIZIONE LOGICA DIGITALE DEI FLIP-FLOP	2-27
	UNA SIMULAZIONE IN LINGUAGGIO ASSEMBLY DEI FLIP-FLOP	2-29
	SIMULAZIONE AL MICROCALCOLATORE DEL FLIP-FLOP IN GENERALE	2-30
	LA SIMULAZIONE AL MICROCALCOLATORE DEI DISPOSITIVI IN TEMPO REALE	2-31
	IL MULTIVIBRATORE MONOSTABILE 555	2-31
	IL MULTIVIBRATORE MONOSTABILE 74121	2-32
	IL FLIP-FLOP 74107 DUALE J-K MASTER-SLAVE CON CLEAR	2-34
	SIMULAZIONE AL MICROCALCOLATORE IN TEMPO REALE	2-35
	CICLI DI ISTRUZIONI DI TIMING DEL MICROCALCOLATORE	2-35

SOMMARIO

Capitolo		Pagina
	I LIMITI DELLA SIMULAZIONE DELLA LOGICA DIGITALE	2-39
	REALIZZAZIONE DELL'INTERFACCIA CON ONE-SHOTS ESTERNI	2-39
	TIME OUT ED INTERRUPTS	2-41
3	UNA SIMULAZIONE DIRETTA DELLA LOGICA DIGITALE	3-1
	COME FUNZIONA LA STAMPANTE QUME	3-2
	SEGNALI D'INGRESSO ED USCITA	3-8
	DISPOSITIVI INGRESSO/USCITA	3-8
	L'INTERFACCIA PERIFERICA PROGRAMMABILE 8255	3-9
	LA PORTA INGRESSO/USCITA AD 8-BIT 8212	3-12
	$\overline{\text{RETURN STROBE}}$	3-13
	SEGNALI D'INGRESSO	3-13
	PFL REL	3-14
	RIB LIFT RDY	3-15
	PW STROBE	3-15
	FFA	3-15
	RESET	3-16
	PFR REL	3-16
	CA REL	3-16
	$\overline{\text{FFI}}$	3-17
	$\overline{\text{EOR DET}}$	3-17
	HAMMER ENABLE FF	3-18
	CLK	3-19
	H1-H6	3-19
	SOMMARIO SUL SEGNALE D'INGRESSO	3-19
	SEGNALE D'USCITA	3-20
	UNA SIMULAZIONE ORIENTATA ALLA LOGICA DIGITALE	3-20
	UN'ANALISI DELLA LOGICA	3-20
	FLIP-FLOP FFA_W	3-22
	SIMULAZIONE DEL FLIP-FLOP FFA_W	3-24
	FLIP-FLOP FFB_W	3-30
	SIMULAZIONE DEL FLIP-FLOP FFB	3-32
	FLIP-FLOP FFC	3-36
	SIMULAZIONE DEL FLIP-FLOP FFC	3-38
	SIMULAZIONE DELL'IMPULSO DI INIZIO MOVIMENTO DEL NASTRO	3-40
	FLIP-FLOP FFD	3-42
	SIMULAZIONE DEL FLIP—FLOP FFD	3-42
	FLIP-FLOP FFE	3-44
	ONE-SHOT PW SETTLING	3-46
	SIMULAZIONE DELLO ONE-SHOT PW SETTLING	3-47
	FLIP-FLOP FFF	3-47
	SIMULAZIONE DEL FLIP-FLOP FFF	3-49
	IL MULTIVIBRATORE 555	3-52
	SIMULAZIONE DEL MULTIVIBRATORE 555	3-52
	IL FLIP-FLOP PW RELEASE ENABLE	3-59

SOMMARIO (continua)

Capitolo		Pagina
	SIMULAZIONE DEL FLIP-FLOP PW RELEASE ENABLE	3-59
	SIMULAZIONE DELLO ONE-SHOT PW READY ENABLE	3-61
	SOMMARIO DELLA SIMULAZIONE	3-64
4	UN SEMPLICE PROGRAMMA	4-1
	TIMING DEL LINGUAGGIO ASSEMBLY IN FUNZIONE DEL TIMING DELLA LOGICA DIGITALE	4-1
	SEGNALI D'INGRESSO E D'USCITA	4-1
	CONFIGURAZIONE DEL DISPOSITIVO A MICROCALCOLATORE	4-3
	CONCETTI DI PROGETTO GENERALE	4-3
	INTERFACCIA PERIFERICA PROGRAMMABILE (PPI) 8255	4-4
	INIZIALIZZAZIONE DEL SISTEMA	4-6
	MEMORIA ROM E RAM	4-7
	DIAGRAMMA DI FLUSSO DEL PROGRAMMA	4-9
	ERRORI DELLA LOGICA DEL PROGRAMMA	4-23
	RESET ED INIZIALIZZAZIONE	4-27
	SOMMARIO DEL PROGRAMMA	4-27
5	UNA PROSPETTIVA DEL PROGRAMMATORE	5-1
	EFFICIENZA DI PROGRAMMAZIONE SEMPLICE	5-1
	CONSULTAZIONI DI TABELLA EFFICIENTE	5-1
	UTILIZZAZIONE DELL'HARDWARE	5-5
	ISTRUZIONI SPECIFICHE-HARDWARE	5-5
	IMPIEGO DIRETTO DELLE CARATTERISTICHE HARDWARE	5-7
	SUBROUTINES	5-9
	CHIAMATA DI SUBROUTINE	5-11
	RITORNO DA SUBROUTINE	5-15
	QUANDO IMPIEGARE LE SUBROUTINES	5-16
	RITORNO CONDIZIONALE DA SUBROUTINE	5-17
	RITORNI MULTIPLI DA SUBROUTINE	5-19
	CHIAMATE DI SUBROUTINE CONDIZIONALI	5-22
	MACROS	5-22
	COS'E' UN MACRO	5-23
	MACRO CON PARAMETRI	5-24
	INTERRUPT	5-27
	CONSIDERAZIONI HARDWARE SULL'INTERRUPT	5-27
	PROGRAMMA DI SERVIZIO INTERRUPT	5-29
	GIUSTIFICAZIONE DEGLI INTERRUPT	5-34
	INTERRUPT MULTIPLI	5-35
6	SET DI ISTRUZIONI PER L'8080/9080	6-1
	ABBREVIAZIONI	6-1
	STATO	6-2
	MNEMONICI DI ISTRUZIONE	6-2
	CODICI OGGETTO DI ISTRUZIONE	6-2

SOMMARIO (continua)

Capitolo		Pagina
	TEMPO DI ESECUZIONE DI ISTRUZIONE E CODICI	6-2
	ACI – SOMMA CON CARRY IMMEDIATO ALL'ACCUMULATORE	6-12
	ADC – SOMMA DI UN REGISTRO O MEMORIA CON CARRY NELL'ACCUMULATORE	6-13
	ADD – SOMMA DI REGISTRO O MEMORIA ALL'ACCUMULATORE	6-14
	ADI – SOMMA IMMEDIATA ALL'ACCUMULATORE	6-16
	ANA – AND DI REGISTRO O MEMORIA CON L'ACCUMULATORE	6-17
	ANI – AND IMMEDIATO CON L'ACCUMULATORE	6-18
	CALL – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO	6-19
	CC – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO CARRY E' UGUALE AD 1	6-20
	CM – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN E' UGUALE AD 1	6-20
	CMA – COMPLETA L'ACCUMULATORE	6-21
	CMC – COMPLETA LO STATO CARRY	6-21
	CMP – CONFRONTA REGISTRO E MEMORIA CON L'ACCUMULATORE	6-22
	CNC – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO CARRY E' UGUALE A 0	6-24
	CNZ – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO ZERO E' UGUALE A 0	6-24
	CP – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN E' UGUALE A 0	6-24
	CPE – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO PARITA' E' UGUALE AD 1	6-25
	CPI – CONFRONTA I CONTENUTI DELL'ACCUMULATORE CON DATI IMMEDIATI	6-26
	CPO – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO DI PARITA' E' UGUALE A 0	6-26
	CZ – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO ZERO E' UGUALE AD 1	6-27
	DAA – AGGIUSTA DECIMALI ACCUMULATORE	6-28
	DAD – SOMMA UNA COPPIA DI REGISTRI AD H ED L	6-29
	DCR – DECREMENTA I CONTENUTI DI REGISTRO DI MEMORIA	6-30
	DCX – DECREMENTA UNA COPPIA DI REGISTRI	6-32
	DI – DISABILITA INTERRUPT	6-33

SOMMARIO (continua)

Capitolo		Pagina
EI	– ABILITA INTERRUPT	6-33
HLT	– ARRESTO	6-35
IN	– INGRESSO ALL'ACCUMULATORE	6-36
INR	– INCREMENTA REGISTRO O CONTENUTI DI MEMORIA	6-36
INX	– INCREMENTA COPPIA DI REGISTRI	6-38
JC	– SALTA SE CARRY	6-39
JM	– SALTA SE MENO	6-39
JMP	– SALTA ALL'ISTRUZIONE IDENTIFICATA NEL-L'OPERANDO	6-40
JNC	– SALTA SE NON CARRY	6-40
JNZ	– SALTA SE NON ZERO	6-41
JP	– SALTA SE POSITIVO	6-41
JPE	– SALTA SE PARITA' PARI	6-41
JPO	– SALTA SE PARITA' DISPARI	6-42
JZ	– SALTA SE ZERO	6-42
LDA	– CARICA L'ACCUMULATORE DELLA MEMORIA UTILIZZANDO L'INDIRIZZAMENTO DIRETTO	6-43
LDAX	– CARICA L'ACCUMULATORE DALLA LOCAZIONE DI MEMORIA INDIRIZZATA MEDIANTE UNA COPPIA DI REGISTRI	6-44
LHLD	– CARICA DIRETTAMENTE I REGISTRI H ED L	6-45
LXI	– CARICA UN VALORE A 16 BIT, IMMEDIATO IN UNA COPPIA DI REGISTRI	6-46
MOV	– MUOVI DATI	6-46
MVI	– MUOVI DATI IN MODO IMMEDIATO IN UN REGISTRO O IN MEMORIA	6-48
NOP	– NON OPERARE	6-50
ORA	– OR DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE	6-51
ORI	– OR IMMEDIATO CON L'ACCUMULATORE	6-52
OUT	– USCITA DALL'ACCUMULATORE	6-53
PCHL	– SALTA ALL'INDIRIZZO SPECIFICATO MEDIANTE HL	6-54
POP	– LEGGI DALLA SOMMITA' DELLO STACK	6-55
PUSH	– SCRIVI NELLA SOMMITA' DELLO STACK	6-56
RAL	– RUOTA L'ACCUMULATORE A SINISTRA ATTRAVERSO IL CARRY	6-57
RAR	– RUOTA L'ACCUMULATORE A DESTRA ATTRAVERSO IL CARRY	6-58
RC	– RITORNA SE LO STATO CARRY E' UGUALE AD 1	6-59
RET	– RITORNA DA SUBROUTINE	6-59
RLC	– RUOTA L'ACCUMULATORE A SINISTRA	6-60
RM	– RITORNA SE LO STATO SIGN E' UGUALE AD 1	6-60
RNC	– RITORNA SE LO STATO CARRY E' UGUALE A 0	6-61
RNZ	– RITORNA SE LO STATO ZERO E' UGUALE A 0	6-62
RP	– RITORNA SE LO STATO SIGN E' UGUALE A 0	6-62

SOMMARIO (continua)

Capitolo		Pagina
	RPE – RITORNA SE LO STATO PARITA' E' UGUALE AD 1	6-63
	RPO – RITORNA SE LO STATO PARITA' E' UGUALE A 0	6-64
	RRC – RUOTA L'ACCUMULATORE A DESTRA	6-64
	RST – RESTART (RIPARTI)	6-65
	RZ – RITORNA SE LO STATO ZERO E' UGUALE AD 1	6-65
	SBB – SOTTRAI UN REGISTRO O MEMORIA DALLO ACCUMULATORE CON PRESTITO	6-66
	SBI – SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE CON PRESTITO	6-68
	SHLD – MEMORIZZA DIRETTO I REGISTRI H ED L	6-69
	SPHL – CARICA IL PUNTATORE DELLO STACK DAI REGISTRI H ED L	6-69
	STA – IMMAGAZZINA L'ACCUMULATORE IN MEMORIA USANDO L'INDIRIZZAMENTO DIRETTO	6-70
	STAX – IMMAGAZZINA IL CONTENUTO DELL'ACCUMULATORE NELLA LOCAZIONE DI MEMORIA INDIRIZZATA DA UNA COPPIA DI REGISTRI	6-71
	STC – PONI AD 1 LO STATO CARRY	6-72
	SUB – SOTTRAI IL CONTENUTO DI UN REGISTRO O MEMORIA DALL'ACCUMULATORE	6-72
	SUI – SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE	6-74
	XCHG – SCAMBIA I CONTENUTI DEI REGISTRI DE ED HL	6-75
	XRA – OR ESCLUSIVO DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE	6-76
	XRI – OR ESCLUSIVO IMMEDIATO DI DATI CON L'ACCUMULATORE	6-77
	XTHL – SCAMBIA LA SOMMITA' DELLO STACK CON HL	6-78
7	ALCUNE SUBROUTINE IMPIEGATE COMUNEMENTE	7-1
	INDIRIZZAMENTO DELLA MEMORIA	7-1
	AUTO-INCREMENTO ED AUTO-DECREMENTO	7-1
	INDIRIZZAMENTO CON INDICE	7-3
	INDIRIZZAMENTO INDIRETTO	7-3
	INDIRIZZAMENTO INDIRETTO, A POST-INDICE	7-4
	MOVIMENTO DATI	7-5
	MOVIMENTO DI BLOCCHI SEMPLICI DI DATI	7-5
	CONSULTAZIONE DI TABELLA MULTIPLA	7-5
	CLASSIFICAZIONE DI DATI	7-5
	ARITMETICA	7-8
	ADDIZIONE BINARIA	7-9
	SOTTRAZIONE BINARIA	7-10
	ADDIZIONE DECIMALE	7-11

SOMMARIO (continua)

Capitolo		Pagina
	SOTTRAZIONE DECIMALE	7-11
	MOLTIPLICAZIONE E DIVISIONE	7-12
	MOLTIPLICAZIONE BINARIA AD 8-BIT	7-12
	MOLTIPLICAZIONE BINARIA A 16-BIT	7-16
	DIVISIONE BINARIA	7-16
	LOGICA DELLA SEQUENZA DI ESECUZIONE DEL PROGRAMMA	7-17
	LA TABELLA DI SALTO	7-17
APPENDICE		
A	CODICI DI CARATTERE ASCII	A-1

INDICE DELLE FIGURE

Figura		Pagina
3-1	Logica di controllo della ruota di stampa	—
3-2	Diagramma del timing della logica di controllo della ruota di stampa	3-4
3-3	Il programma di simulazione completa	3-66
4-1	Timing per la Figura 3-1 dal punto di vista del programmatore	4-2
4-2	Configurazione del microcalcolatore	4-5
4-3	Primo tentativo di costruzione del diagramma di flusso del programma	4-8
4-4	Diagramma di flusso del programma per calcolare la lunghezza dell'impulso di alimentazione del martelletto	4-16
4-5	Una semplice sequenza d'istruzione del ciclo senza inizializzazione o reset	4-18
4-6	Un semplice programma del ciclo di stampa	4-28
5-1	Configurazione del microcalcolatore con un interrupt singolo	5-26
5-2	Generazione del codice dell'istruzione restart seguente un interrupt, per un sistema a microcalcolatore 8080	5-36

INDICE DELLE TABELLE

Tabella		Pagina
2-1	Confronto della utilizzazione di memoria e della velocità di esecuzione di un programma per la simulazione delle porte AND 7411	2-26
5-1	La lunghezza di subroutine conveniente più corta in funzione del numero di volte di chiamata della subroutine stessa	5-17
6-1	Sommario del Set di istruzioni del Microcalcolatore 8080/9080	6-3
6-2	Sommario dei codici oggetto delle istruzioni e dei cicli di esecuzione.	6-11

INDICE ANALITICO

Indice		Pagina
A	ABILITAZIONE DI INTERRUPT	5-27
	AMPLIFICATORE	2-14
	ASSEGNAZIONE DI LABEL DEL PROGRAMMA SORGENTE	2-21
	ASSEGNAZIONE DEI PIN	4-3
B	BIT DATI	2-5, 2-6
	BREVI INTERVALLI DI TEMPO DI TIMING	2-35
	BUFFER	2-14
	BUFFER DI TIPO LATCH	3-8
C	CALCOLO DELLA LARGHEZZA D'IMPULSO	3-41
	CALCOLO DEL RITARDO DI TEMPO	3-59
	CALCOLO DELL'INDIRIZZO DELLA MEMORIA DATI	3-57
	CAMBIAMENTI DELLO STATO CON L'ESECUZIONE DI ISTRUZIONI	6-2
	CARICAMENTO DI INDIRIZZI NEL PUNTATORE DELLO STACK	7-2
	CARICHI TTL	2-14
	CH RDY	3-5
	CHIAMATA DI SUBROUTINE USANDO RST	6-65
	CICLO DI PROGETTO IN LOGICA DIGITALE	2-1
	CICLO DI PROGETTO LOGICO CON MICROCALCOLATORI	2-2
	CLASSIFICA DELLE VARIAZIONI DI PROGRAMMA	2-27
	CLEAR DEL FLIP-FLOP	2-29
	CODICE DI CONTROLLO PPI	5-8
	COMMUTAZIONE OFF DI BIT	3-32
	COMMUTAZIONE ON DI BIT	3-32
	COMMUTAZIONE ON DI UN BIT	3-27
	COMPLEMENTAZIONE DI UN BYTE DI MEMORIA	2-15
	CONDIZIONI DI STATO	6-57
	CONFIGURAZIONE DI INTERRUPT SINGOLO	5-28
	CONFLITTI NELL'UTILIZZAZIONE DEI REGISTRI DELLA CPU	2-24
	CONFRONTO IMMEDIATO	4-25
	CONFRONTO INDIRIZZAMENTO DIRETTO ED IMPLICATO	2-26
	CONSERVAZIONE DEL PUNTATORE DELLO STACK	7-2
	CONSERVAZIONE DEI REGISTRI E DELLO STATO	5-32
	CONSIDERAZIONI DI TIMING DI INTERRUPT	5-34
	CONTROLLO DI LIMITE	4-23
	CONVALIDA DI STATO IMPIEGANDO L'ISTRUZIONE DCX	2-36
	CORRENTE DI LEAKAGE	2-14
	COSTRUZIONE DEL CODICE D'ISTRUZIONE RST	5-28
D	DECODIFICA DEL BUS INDIRIZZI	3-11
	DEFINIZIONE DI MACRO	5-23
	DETERMINAZIONE DELL'INDIRIZZO DELLA PORTA I/O	3-10

INDICE ANALITICO (continua)

Indice		Pagina
	DETERMINAZIONE DELLO STATO OPERANDO L'AND DI UN REGISTRO CON SE' STESSO	2-18
	DIAGRAMMA DI FLUSSO	2-5
	DIRETTIVE DELL'ASSEMBLER MACRO	5-23
E	ECONOMIA DELL'INTERRUPT	5-34
	ESECUZIONE DI PROGRAMMI ALL'INTERNO DI RITARDI DI TEMPO	2-37
F	FAN IN	2-14
	FAN IN NEI PROGRAMMI DEL MICROCALCOLATORE	2-17
	FAN OUT	2-14
	FAN OUT NEI PROGRAMMI DI MICROCALCOLATORE	2-19
	FF A	3-7
	FLAGS DI STATO IMPIEGATI PER RAPPRESENTARE LA LOGICA	3-25
	FLIP-FLOP DI TIPO D	2-28
	FLIP—FLOP MASTER—SLAVE	2-31,2-34
	FLIP-FLOP JK	2-28
	FLIP-FLOP 7474	3-22
	FUNZIONE DI TRASFERIMENTO	4-1
H	H NEL CAMPO DELL'OPERANDO	2-11
I	IMPIEGO DELL'INDIRIZZAMENTO DI MEMORIA IMPLICATO	2-22
	IMPULSO D'INIZIO DEL NASTRO	3-7
	IMPULSO DI SEGNALE PROGRAMMATO	4-20
	IMPULSO VARIABILE ONE—SHOT	3-52
	INDIRIZZAMENTO IMPLICATO	2-25
	INDIRIZZAMENTO DELLA PORTA I/O	3-10
	INDIRIZZAMENTO DI MEMORIA MEDIANTE IL PUNTATATORE DELLO STACK	7-1
	INDIRIZZI DI MEMORIA	4-7
	INGRESSO/USCITA	2-7
	INIZIALIZZAZIONE DEL RITARDO DI TEMPO	2-37
	INIZIALIZZAZIONE ONE-SHOT	2-39
	INTERFACCIA PERIFERICA PROGRAMMABILE	3-8
	INTERPRETAZIONE DEL CODICE OGGETTO	2-8,2-10
	I/O CON STROBE	3-10
	I/O NELLO SPAZIO DI INDIRIZZO DI MEMORIA	2-7
	I/O SEMPLICE	3-9
	ISTRUZIONE DI RESTART	5-27
	ISTRUZIONI DIPENDENTI HARDWARE	5-6
	ISTRUZIONI DI SET/RESET DI BIT	5-5
L	LARGHEZZA DELL'IMPULSO	3-16
	LARGHEZZA DELL'IMPULSO DEL SEGNALE DI INGRESSO	3-15

INDICE ANALITICO (continua)

Indice		Pagina
	LINGUAGGI AD ALTO LIVELLO	4-4
	LINGUAGGIO ASSEMBLY RISPETTO ALLA LOGICA DIGITALE	3-65
	LOCAZIONE DI DEFINIZIONE MACRO IN UN PROGRAMMA SORGENTE	5-24
	LOGICA ASINCRONA	2-11
	LOGICA COMBINATORIA	1-1
	LOGICA ESCLUSA DAL MICROCALCOLATORE	3-52
	LOGICA DI RESET	4-6
	LOGICA ESTERNA COME SORGENTE O DESTINAZIONE	2-7
	LOGICA SINCRONA	2-11
	LUNGHI INTERVALLI DI TEMPO DI TIMING	2-36
M	MANIPOLAZIONE DELLO STACK	5-21
	MASCHERATURA DI BIT	2-10
	MODI DELLA PORTA I/O	3-9
	MODO DI SELEZIONE DELLA PORTA I/O	3-10
	MULTIVIBRATORE MONOSTABILE	2-31
O	ONE-SHOT	2-31
	ONE-SHOT TIME OUT IMPIEGANDO LO STATO	2-40
	ORIGINE DEL PROGRAMMA DI INTERRUPT	5-30
P	PARAMETRO DI SUBROUTINE	5-18
	PERDITA DI TEMPO DEL SERVIZIO INTERRUPT	5-34
	PROTA I/O 8212 IMPIEGATA IN UN SISTEMA D'INTERRUPT	5-28
	PORTE I/O INDIRIZZATE COME MEMORIA	3-12
	POSIZIONE DI VISIBILITA' DELLA RUOTA DI STAMPA	3-7
	PRESET DEL FLIP-FLOP	2-29
	PROGRAMMA OGGETTO	2-4
	PROGRAMMA SORGNETE	2-4
	PROGRAMMI RESI PIU' BREVI	3-40, 3-43
	PW STROBE	3-5
Q	QUANDO IMPIEGARE GLI INTERRUPTS	5-27
R	RAM	4-7
	REGISTRI DELLA CPU	2-5
	RESET	3-24
	RESET DELLA CPU	3-16
	RICONOSCIMENTO DI INTERRUPT	5-27
	RI-IMMAGAZZINAMENTO DEL PUNTATORE DELLO STACK	7-2
	RIPOSIZIONAMENTO DELLA RUOTA DI STAMPA PER IL CICLO DI STAMPA	3-31
	RIPOSIZIONAMENTO DELLA RUOTA PER IL CICLO DI STAMPA	3-13
	RITARDI DI POSIZIONAMENTO	3-6

INDICE ANALITICO (continua)

Indice		Pagina
	RITARDI DI TEMPO SIMULTANEI	2-39
	RITARDO DI ALIMENTAZIONE DEL MARTELLETTO	4-15
	RITARDO DI TEMPO	3-59
	RITARDO DI TEMPO BASATO SUL SEGNALE D'INGRESSO	3-17
	RITARDO DI TEMPO DI LUNGHEZZA VARIABILE	3-45, 4-20
	RITORNO CONDIZIONALE	5-18
	RIVELAZIONE DI CAMBIAMENTI DI LIVELLO DI SEGNALE SENZA INTERRUPTS	3-27
	ROM	4-7
	RUOTA DI STAMPA PRONTA	3-5
S	SALTA SU NO CARRY	3-26
	SALTO SU CONDIZIONE	4-25
	SCELTA DEI DATI	7-7
	SEGNALE ENABLE	3-54
	SEGNALE DI CLOCK	2-28
	SEGNALI D'INGRESSO	4-1
	SELEZIONE DELLA PORTA I/O	4-
	SELEZIONE DEL CHIP IN SISTEMI PIU' GRANDI	4-6
	SELEZIONE DEL CHIP IN SISTEMI SEMPLICI	4-6
	SELEZIONE DEL PIN DELLA PORTA I/O	2-10
	SELEZIONE DI ROM IN SISTEMI SEMPLICI	4-7
	SET/RESET DI BIT ILLUSTRATO	5-7
	SEQUENZA DI EVENTO	3-54
	SEQUENZA DI ISTRUZIONE PER LA SELEZIONE DEL MODO DELLA PORTA I/O	3-11
	SEQUENZA DI REALIZZAZIONE DEL PROGRAMMA	4-6
	SIMULAZIONE DELL'INVERTITORE	3-25
	SIMULAZIONE DEL FLIP-FLOP IMPIEGANDO PORTE I/O	3-24
	SIMULAZIONE DEL GATE OR	3-25
	SIMULAZIONE DEL RITARDO DI TEMPO ONE-SHOT	3-47
	SOMMITA' DELLO STACK	5-30
	SORGENTE E DESTINAZIONE DEI DATI	2-5
	SUBROUTINE IDENTIFICATE	5-18
	STATO CARRY	3-26
	STATO ZERO	3-26
T	TABELLA POSIZIONATA PER SEMPLIFICARE LA SEQUENZA D'ISTRUZIONE DI ACCESSO	5-4
	TEMPO DI RITARDO DI GATE	2-12
	TEMPORIZZAZIONE DEGLI EVENTI IN UN SISTEMA A MICROCALCOLATORE	3-27
	TIMING E LIMITI DI SIMULAZIONE	3-43
	TIMING E SEQUENZA LOGICA	3-28,3-35,3-39
	TIMING DEL PROGRAMMA	2-5
	TRAIETTORIE DI ESECUZIONE DI ISTRUZIONE CONDIZIONALE	4-26
	TRIGGER A GRADINO NEGATIVO	2-28
	TRIGGER A GRADINO POSITIVO	2-27
V	VIA I/O PORTE I/O	2-9

Capitolo 1

INTRODUZIONE

LOGICA COMBINATORIA

Questo libro spiega come un programma in linguaggio assembly, dentro un sistema a microcalcolatore, possa sostituire una logica combinatoria — che è l'uso combinato di "off-the-shelf", dispositivi a logica non programmabile, come la logica digitale della serie standard 7400.

Questo libro si propone di insegnare ai progettisti logici come eseguire in un modo nuovo un vecchio lavoro — mediante la creazione di programmi in linguaggio assembly, all'interno di un sistema a microcalcolatore.

Questo libro si propone di mostrare ai programmatori come la programmazione abbia trovato uno scopo nuovo nel progetto logico.

Questo è un libro "how to do it"; come tale esso diventa molto specifico e ci si riferisce direttamente ad un particolare tipo di microcalcolatore, l'8080A. Un certo numero di società costruisce il microcalcolatore del tipo 8080A; specificamente questi prodotti comprendono:

AMA	**9080A**
INTEL	**8080A**
NEC	**8080A**
TMS	**8080A**
NS	**8080A**

Le società che costruiscono questi microcalcolatori sono:

INTEL CORPORATION
3065 Bowers Avenue
Santa Clara, California 95051

ADVANCE MICRO DEVICES
901 Thompson Place
Sunnyvale, California 94086

TEXAS INSTRUMENTS, INC.
P.O. BOX 1444
Houston, Texas 77001

NEC MICROCOMPUTERS, INC.
5 Militia Drive
Lexington, Massachusetts 02173

NATIONAL SEMICONDUCTOR CORP.
2900 Semiconductor Drive
Santa Clara, California 95050

CHE COSA SI PRESUME NOTO PER LA LETTURA DI QUESTO LIBRO

Questo libro è un seguito di "An Introduction To Microcomputers", che si compone di un unico volume nella sua prima edizione e di due nella seconda.

"An Introduction To Microcomputers" descrive concettualmente i microprocessori ed i microcalcolatori; esso non indirizza alla materia pratica ed alla realizzazione dei concetti. Questo libro al contrario indirizza alla pratica di realizzazione.

Questo libro è un seguito nel senso che esso fa l'assunzione che sia stato letto ovvero altrimenti compreso il materiale esposto in "An Introduction To Microcomputers". Comunque prima di iniziare un vero disegno di progetto si richiederà al venditore la letteratura che descrive specificamente i dispositivi che si desidera impiegare.

In particolare si noti che in questo libro non vengono descritti l'hardware ed il timing per le CPU 8080A/9080A o di qualunque altro dispositivo a microcalcolatore; le informazioni sufficienti possono essere reperite in "An Introduction To Microcomputers", Volume II, Some Real Products.

Il set di istruzione dell'8080A è descritto al Capitolo 6 di questo libro, poichè questo libro si occupa interamente di programmazione.

COMPRENSIONE DEL LINGUAGGIO ASSEMBLY

Le istruzioni in linguaggio assembly sono le funzioni di trasferimento di un sistema a microcalcolatore; considerate assieme esse costituiscono un "set di istruzioni" che descrive le singole operazioni che il microcalcolatore può eseguire.

Si può definire l'evento, che deve ricorrere dentro il sistema seriale del microcalcotore — come la sequenza di istruzioni che, considerate assieme, costituiscono un programma in linguaggio assembly.

In realtà la comprensione del significato di ogni singola istruzione all'interno del sistema a microcalcolatore è molto comoda; questo è uno degli aspetti più semplici dell'elaborazione con i microcalcolatori. Questo spaventa ancora gli utenti in modo eccessivo, particolarmente se non esperti di programmazione.
A questi ultimi và rivolta una parola di raccomandazione — non si considerino i mnemonici ed i set di istruzione; si considerino le istruzioni una alla volta, come vengono incontrate in questo libro. Quando non si comprende l'esecuzione di una istruzione la si consulti al Capitolo 6.

Lo spettro della "programmazione" perseguiterà solo chi lo consente.

FORMA DELLA STAMPA DI QUESTO LIBRO

Si noti che il testo di questo libro è stato stampato in caratteri in grassetto alternati a caratteri normali. **Questo è stato fatto per aiutare ad evitare quelle parti del libro con cui si ha già familiarità. Infatti i caratteri normali sono soltanto una espansione dell'informazione presentata nei precedenti caratteri in grassetto.** Perciò si leggano solo i caratteri in grassetto finchè non si reperisce un soggetto del quale si vuole conoscere di più ed a questo punto si inizi la lettura dei caratteri normali.

Capitolo 2

LINGUAGGIO ASSEMBLY E LOGICA DIGITALE

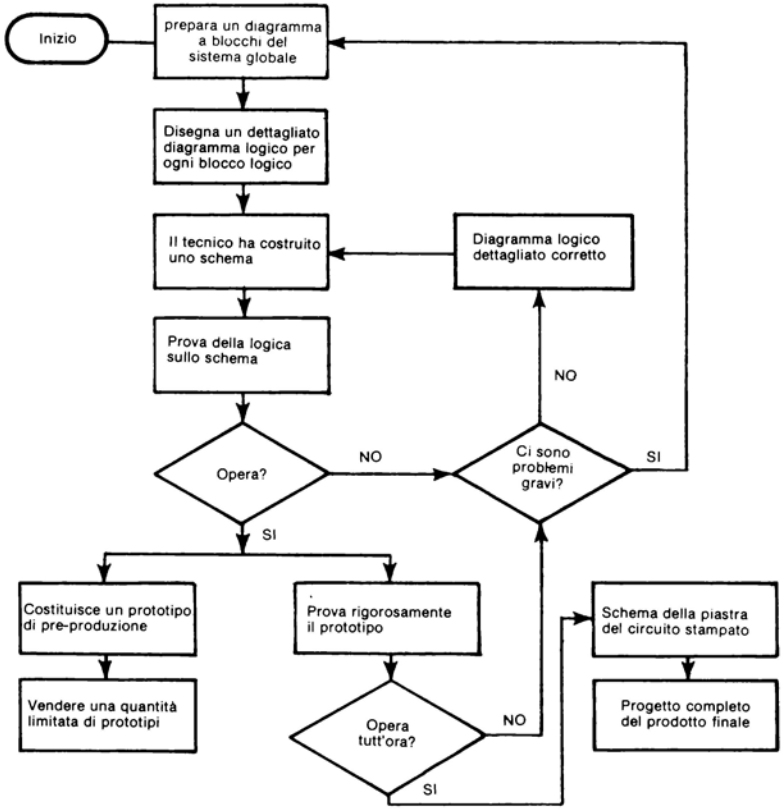
IL CICLO DI PROGETTO

CICLO DI PROGETTO IN LOGICA DIGITALE

Qualsiasi prodotto che è costituito al di fuori dei componenti della logica digitale passerà attraverso un ben definito ciclo di progetto.

Si assumerà che i prodotti siano stati definiti dal punto di vista della gestione marketing.

La specifica del prodotto identifica necessariamente le caratteristiche e le esecuzioni del prodotto; il lavoro dell'utente coinvolge necessariamente il percorso di progetto del costruttore. **Il ciclo di progetto procederà come segue:**



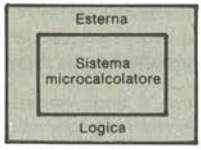
Nel ciclo di qualunque progetto logico digitale esiste un anello iterativo lento e dispendioso; come illustrato esso consiste di queste fasi:

- Schizzo logico
- Costruzione di un nuovo schema
- Controllo dello schema da errori di tipo logico, errori tecnici e componenti non corretti.

Questo ciclo iterativo rende il progetto logico combinatorio lento e dispendioso — non solo durante la fase iniziale del progetto, ma anche quando si decide successivamente di modificare od aumentare il prodotto.

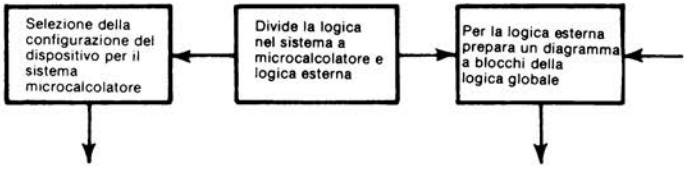
CICLO DI PROGETTO LOGICO CON MICROCALCOLATORI

Cosa succede quando si inizia ad impiegare i microcalcolatori? Prima di tutto una porzione di logica svanisce in una "scatola nera" che è il sistema microcalcolatore:



La prima fase:

deve essere spezzata come segue:

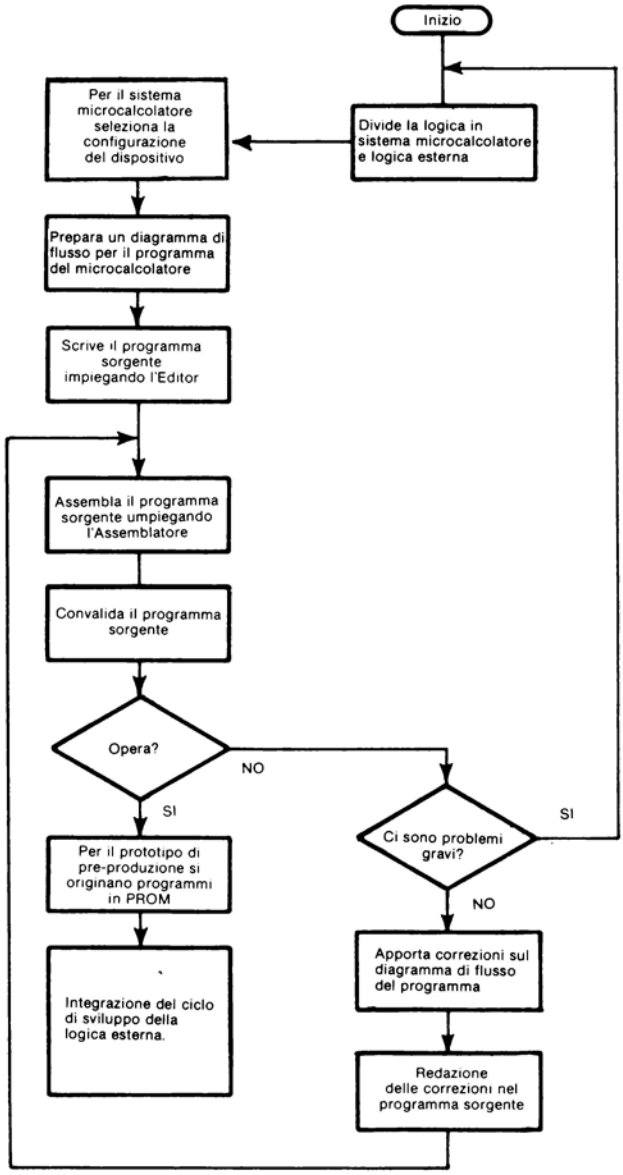


La ripartizione dell'applicazione in esame in un sistema a microcalcolatore ed a logica digitale esterna può essere considerata un difficile traguando — se non si comprende cosa può fare il sistema a microcalcolatore.

Infatti, **avendo a disposizione un microcalcolatore, gli economisti assumeranno in modo deciso che la "scatola nera" può eseguire qualsiasi compito; si deve quindi giustificare l'esistenza di ogni singolo gate logico esterno.**

Si ricordi: la memoria è limitata. Allo scopo di espandere la realizzazione logica all'interno del sistema a microcalcolatore si deve semplicemente scrivere sequenze di istruzioni addizionali che risiederanno in memoria che altrimenti sarebbe sprecata; aggiungendo memoria di programma, per questo scopo il costo è molto basso.

Inoltre, confrontata col costo della logica di sviluppo digitale, lo sviluppo della logica del microcalcolatore è veloce e non dispendiosa. **Un tipico ciclo di sviluppo di sistema a microcalcolatore potrebbe essere illustrato come segue:**



Tuttavia ci sono anelli iterativi nel ciclo di sviluppo illustrato precedentemente che però, confrontati con lo sviluppo della logica digitale, impiegano meno tempo e sono meno dispendiosi.

Ogni microcalcolatore è sostenuto da un sistema di sviluppo. Le caratteristiche di questi sistemi di sviluppo variano notevolmente da una società ad un'altra, comunque essi hanno tutti queste possibilità:

1) Si puo **simulare il sistema a microcalcolatore** configurato senza necessariamente creare i mezzi di sostegno.

PROGRAMMA SORGENTE

2) Si può eseguire un programma editor residente per **creare la sorgente del programma.** Si ricordi che una sequenza di istruzioni in linguaggio assembly costituiscono un "Programma Sorgente".

PROGRAMMA OGGETTO

3) Si può **assemblare il programma sorgente** appropriato al sistema di sviluppo per creare un programma oggetto. Si ricordi il programma sorgente redatto in una sequenza di digits binari viene chiamato programma oggetto, prima che esso venga eseguito.

4) Si può **incondizionatamente eseguire il programma oggetto** per assicurarsi che esso operi correttamente.

Impiegando un tipico sistema di sviluppo a microcalcolatore si possono considerare alcuni tra i cicli di sviluppo in un solo giorno, dove ogni ciclo di sviluppo potrebbe avere occupato una o due settimane per la realizzazione completa della logica digitale. All'interno di un singolo ciclo di sviluppo si possono fare molte correzioni di programma; in meno di un minuto si può fare una semplice correzione, equivalente ad aggiungere o togliere un gate (o funzione MSI) da un sistema logico digitale.

SIMULAZIONE DELLA LOGICA DIGITALE

Quindi la logica deve eventualmente essere separata in sistema a microcalcolatore e logica esterna al sistema a microcalcolatore.

Si vogliono far notare due aspetti di questa separazione logica:

1) Basandosi sull'abilità del linguaggio assembly a simulare la logica digitale, **si deve sviluppare qualche semplice criterio per stimare cosa può fare un sistema a microcalcolatore e cosa non può fare.**

2) **Si deve creare un programma per realizzare le funzioni logiche assegnate al sistema a microcalcolatore.** Sfortunatamente esistono innumerevoli modi di scrivere un programma di un microcalcolatore. Una volta capito il concetto dell'impiego delle istruzioni per guidare un sistema a microcalcolatore, **la fase successiva è di imparare a scrivere programmi efficienti.**

Si comincerà descrivendo una semplice simulazione della logica digitale. Questa parte preliminare è necessaria perchè ci sono alcune differenze concettuali tra la logica digitale e la logica di programmazione del microcalcolatore.

SIMULAZIONE AL MICROCALCOLATORE DI UN INVERTITORE DI SEGNALE

Si supponga di voler invertire un singolo segnale:

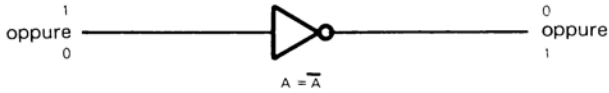


DIAGRAMMA DI FLUSSO

Allo scopo di sviluppare le nuove abitudini dall'inizio si illustrerà l'invertitore di segnale con il seguente diagramma di flusso logico:

Benchè non si userà mai un microcalcolatore semplicemente per sostituire un invertitore di segnale, tuttavia vale la pena esaminare come potrebbe essere realizzato.

UNA SEQUENZA DI EVENTI DI UN MICROCALCOLATORE

REGISTRI DELLA CPU

Si ricorda che i microcalcolatori del tipo 8080 hanno i seguenti registri della CPU:

	A	Accumulatore primario
B	C	Accumulatori secondari/Contatore Dati
D	E	Accumulatori secondari/Contatore Dati
H	L	Accumulatori secondari/Contatore Dati
SP		Puntatore dello Stack
PC		Contatore di Programma

La singola istruzione:

```
CMA          ; Complementa l'Accumulatore
```

BIT DATI

quando è convertita in codice oggetto ed eseguita, inverte tutti gli otto bit dell'Accumulatore primario. Ma questo non uguaglia l'invertitore. Primo, un digit binario dell'Accumulatore deve essere selezionato per rappresentare il segnale invertito. Ma quale?

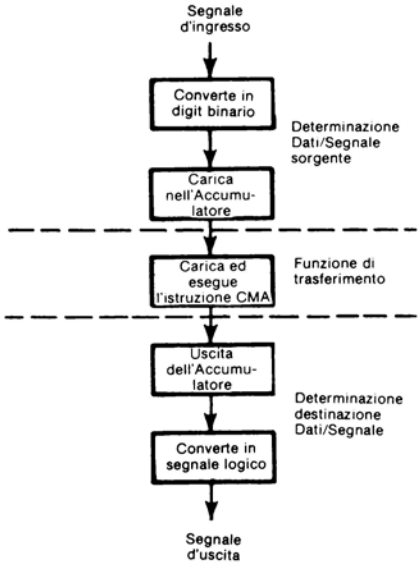
SORGENTE E DESTINAZIONE DEI DATI

Avendo deciso quale digit binario come raggiungere l'Accumulatore nella prima posizione? Una volta invertito come diventa ancora un segnale il bit invertito?

TIMING DEL PROGRAMMA

Se il codice oggetto dell'istruzione CMA deve essere eseguito allo scopo di fornire l'effettiva inversione come e quando il codice oggetto raggiunge la CPU? Chiaramente l'esecuzione di questa istruzione deve essere temporizzata in modo che il digit binario da invertire raggiunga l'Accumulatore.

Le fasi necessarie per realizzare un invertitore impiegante un microcalcolatore possono essere illustrate espandendo il diagramma di flusso come segue:



Nella precedente struttura si faccia molta attenzione alla divisione del problema in queste tre fasi:

1) **Determinazione sorgente dati/segnale.** Si identificano i dati da trattare. Questi dati sono trasferiti ad una locazione alla quale può accadere la Unità di Elaborazione Centrale (CPU) del microcalcolatore.

2) **Esecuzione della funzione di trasferimento.** L'operazione effettiva dovrà essere eseguita sui dati di sorgente e costituirà la "Funzione di Trasferimento".

3) **Determinazione della destinazione dei dati/segnale.** I dati ed i segnali soggetti alla funzione di trasferimento devono ora essere trasferiti ad una particolare destinazione.

Si costruirà ora una sequenza di istruzione per realizzare le tre fasi precedentemente illustrate della simulazione dell'invertitore.

REALIZZAZIONE DELLA FUNZIONE DI TRASFERIMENTO

BIT DATI

L'istruzione CMA inverte ogni bit dell'Accumulatore.

L'istruzioe CMA perciò non specifica quale bit dell'Accumulatore rappresenta il segnale da invertire. Questa specifica è implicata dal modo in cui entrano ed escono i dati dal sistema a microcalcolatore.

DETERMINAZIONE DELLE SORGENTI E DESTINAZIONI DEI DATI

Come i dati dell'Accumulatore entreranno ed usciranno dal sistema a microcalcolatore? Rispondendo a questa domanda si tocca uno dei punti di forza fondamentali (e più complessi) dei microcalcolari — la loro flessibilità.

LA LOGICA ESTERNA COME SORGENTE O DESTINAZIONE

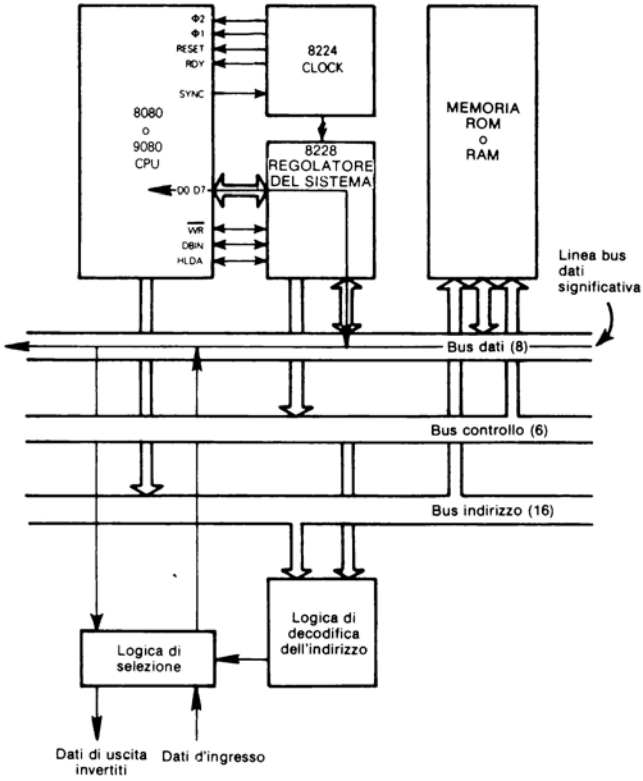
Il segnale d'ingresso ed il segnale d'uscita invertito sono proprio ciò che implica il loro nome — segnali. Ma per il sistema a microcalcolatore essi sono una "logica esterna". I trasferimenti d'informazione tra la logica esterna ed il sistema a microcalcolatore sono generalmente indicati Ingresso/Uscita (oppure I/O).

INGRESSO/ USCITA

Durante qualsiasi operazione programmata I/O si ricordi che il microcalcolatore è il master e la logica esterna è lo slave. Questo significa che il microcalcolatore deve indicare la direzione della operazione I/O (ingresso od uscita) e deve identificare la logica esterna cui si deve accedere.

I/O NELLO SPAZIO DI INDIRIZZO DI MEMORIA

La logica esterna deve decodificare uno specifico indirizzo di memoria come uno strobe di abilitazione cosicchè l'I/O è manipolato come se esso fosse nella memoria di lettura e scrittura. **Si supponga che sia stata impiegata la label INVD nel programma sorgente in linguaggio assembly per identifi-**



care il segnale da invertire. Questa è la sequenza di istruzione che riprodurrà l'invertitore di segnali.

```
LDA     INVD     ; Carica l'Accumulatore da INVD
CMA              ; Complementa l'Accumulatore
STA     INVD     ; Immagazzina i contenuti dell'Accumulatore ad INVD
```

In termini di dispositivi a microcalcolatore, la configurazione impiegata è quella di pag.2-7.

Quando viene eseguita LDA, la "Logica di Decodifica dell'Indirizzo" origina la "Logica Selezionata" per trasmettere il segnale "Dati In" al Bus Dati.

Esistono otto linee di Bus Dati; il numero della linea a cui è connesso il segnale "Dati In" diventa il numero del bit significativo all'interno dell'Accumulatore. Quando è stata completata l'esecuzione dell'istruzione LDA i contenuti del Bus Dati saranno nell'Accumulatore.

Successivamente viene eseguita l'istruzione CMA. Questa istruzione complementa ogni bit dell'Accumulatore.

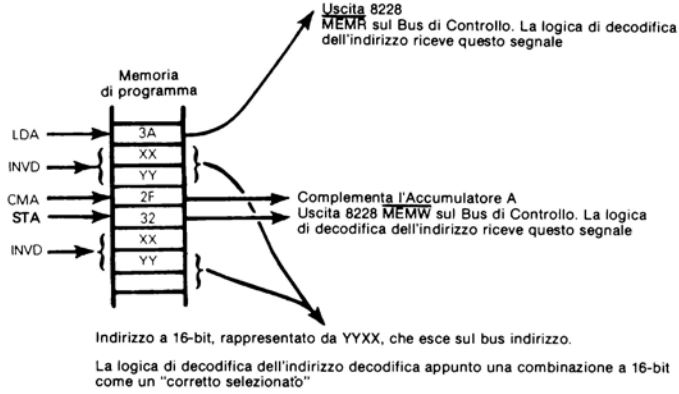
Quando è stata eseguita l'istruzione STA i contenuti dell'Accumulatore sono usciti sul Bus Dati. La "Logica di Decodifica dell'Indirizzo" causa poi la "Logica di Selezione" per fare uscire i contenuti di una singola linea del Bus Dati che diventa il segnale invertito "Dati out".

Poichè la "Logica di Selezione" ha i segnali "Dati In" e "Dati Out" connessi alla stessa linea del Bus Dati, "Dati Out" è il complemento di "Dati In" e l'invertitore di segnale è stato simulato.

La memoria ROM o RAM deve essere presente nel sistema a microcalcolatore poichè il codice oggetto per le tre istruzioni deve essere immagazzinato all'ingresso e portato fuori dalla memoria.

INTERPRETAZIONE DEL CODICE OGGETTO

Si considera in dettaglio il codice oggetto. Le tre istruzioni del programma sorgente diventano codice oggetto come segue:



Gli indirizzi della memoria di programma dei bytes interni che il codice oggetto ha immagazzinato non sono importanti. Comunque nessun byte di memoria ROM o RAM può avere l'indirizzo rappresentato da YYXX poichè la logica esterna è selezionata mediante questo indirizzo.

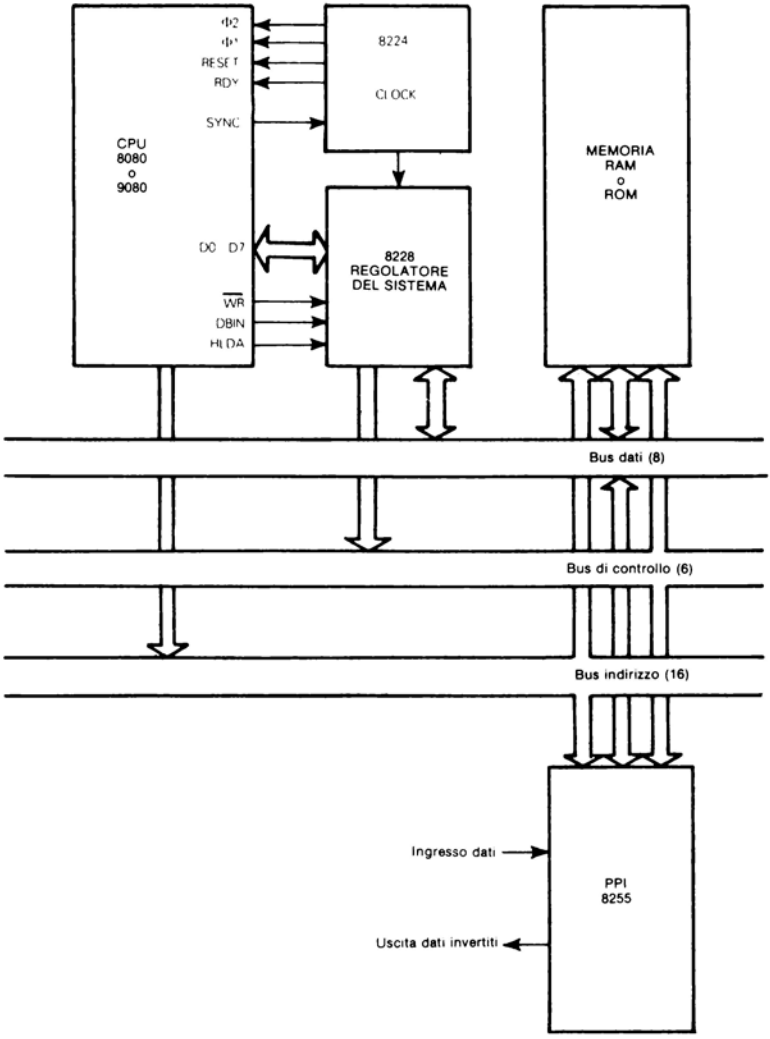
Si osservi i due bytes dell'indirizzo a 16 bit YYXX sono invertiti quando vengono immagazzinati nella memoria. Non c'è nulla di molto significativo a proposito di questa inversione, essa è proprio lo scopo per cui è stato progettato il dispositivo 8080.

VIA I/O PORTE I/O

Si supponga che la comunicazione con la logica esterna avvenga attraverso un dispositivo di interfaccia periferico I/O. Nelle istruzioni del programma sorgente in linguaggio assembly la label INVD identificherà ora una porta I/O. Questa è una sequenza di istruzioni che riproduce lo invertitore di segnale:

```
IN      INVD    ; Ingresso all'Accumulatore dalla porta INVD
CMA             ; Complementa l'Accumulatore
OUT     INVD    ; Uscita dell'Accumulatore alla porta INVD
```

In termini di hardware questa è la configurazione del microcalcolatore:



E' stato fatto tutto aggiungendo l'interfaccia Periferica Programmabile 8255 che fornisce la "Decodifica dell'Indirizzo" e la "Logica di Selezione" necessari dai "Dati In" ed i segnali invertiti "Dati Out". Ora il bit particolare che è significativo sarà determinato mediante il pin del PPI 8255 per cui i segnali "Dati In" e "Dati Out" invertito sono connessi. A loro volta questi pins saranno determinati dal modo in cui viene impiegato l'8255 PPI.

Il fatto che esista un certo numero di scelte disponibili quando si impiega l'8255 PPI è una conseguenza non immediata, che potrà confondere la prima comprensione di tutto quanto coinvolto dalla programmazione in linguaggio assembly. Si ignoreranno perciò le istruzioni di controllo di modo opportuno selezionato.

INTERPRETAZIONE DEL CODICE OGGETTO

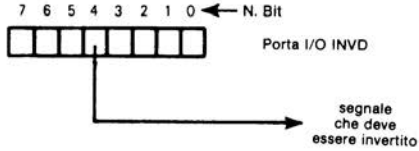
In questo caso il codice oggetto per le tre istruzioni è interpretato come segue:

Ancora una volta gli indirizzi dei bytes della memoria di programma, all'interno dei quali è immagazzinato il precedente codice oggetto, non saranno importanti.

SELEZIONE DEL PIN DELLA PORTA I/O

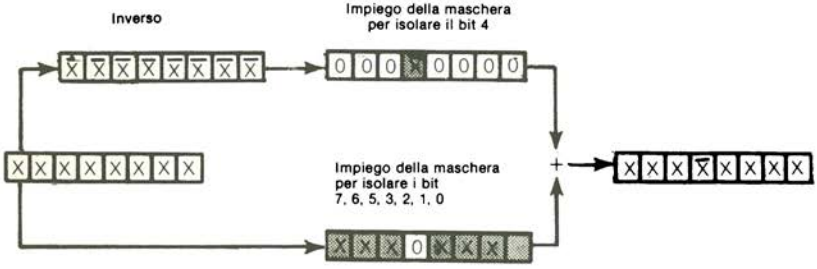
Si osservi che si sta completando ogni bit della porta I/O INVD, anche se solo un bit corrisponde al segnale invertito.

Si supponga che debba essere invertito solo il bit 4:



MASCHERATURA DI BIT

Si può impiegare la tecnica nota come "mascheratura" allo scopo di invertire un singolo pin della porta I/O, lasciando invariati tutti gli altri. In questo caso la mascheratura può essere illustrata come segue:



Nell'istruzione precedente X rappresenta qualunque digit binario ed $\overline{X}$ il suo complemento.

La seguente sequenza d'istruzione invertirà il pin 4, lasciando inalterati tutti gli altri:

```
IN      INVD    ; Ingresso all'Accumulatore dalla Porta I/O INVD
CMA             ; Complementa l'Accumulatore
ANI     10H     ; Isola il bit 4
MOV     B,A     ; Conserva nel registro B
IN      INVD    ; Ingresso all'Accumulatore dalla porta I/O INVD
ANI     EFH     ; Azzera il bit 4
ORA     B       ; OR di A e B
OUT     INVD    ; Uscita dell'Accumulatore alla porta I/O INVD
```

H NEL CAMPO DELL'OPERANDO

H, come ultimo carattere nel campo dell'operando, specifica un valore dati esadecimali, immediato. Così EFH rappresenta il valore binario:

$$\underbrace{1\ 1\ 1\ 0}_{E}\quad \underbrace{1\ 1\ 1\ 1}_{F}$$

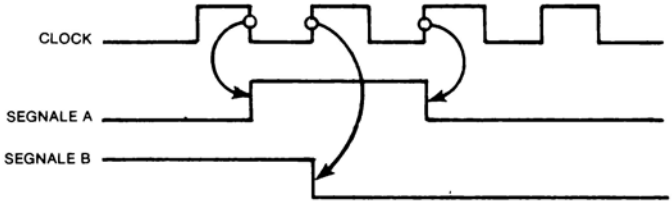
In termini di contenuto di registro, questo è cosa succede quando viene eseguita la precedente sequenza d'istruzione (anche qui X rappresenta qualsiasi digit binario):

		Porta I/O	Accumulatore	Registro B
		XXXXXXXX		?
IN	INVD	XXXXXXXX →	XXXXXXXX	?
CMA		XXXXXXXX	$\overline{X}\overline{X}\overline{X}\overline{X}\overline{X}\overline{X}\overline{X}\overline{X}$	?
			• 00010000	
ANI	10H	XXXXXXXX	000$\overline{X}$0000	?
MOV	B,A	XXXXXXXX	000$\overline{X}$0000 →	000$\overline{X}$0000
IN	INVD	XXXXXXXX →	XXXXXXXX	000$\overline{X}$0000
			• 11101111	
ANI	EFH	XXXXXXXX	XXX0XXXX	000$\overline{X}$0000
			+ 000$\overline{X}$0000	
ORA	B	XXXXXXXX	XXX$\overline{X}$XXXX	000$\overline{X}$0000
OUT	INVD	XXX$\overline{X}$XXXX ←	XXX$\overline{X}$XXXX	000$\overline{X}$0000

TIMING DI EVENTO

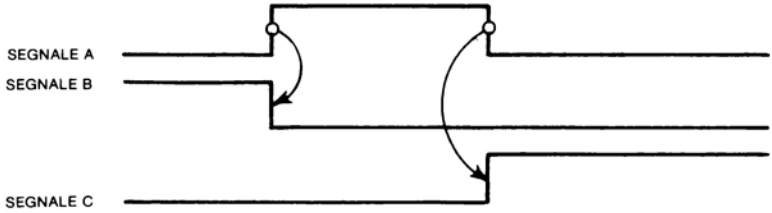
LOGICA SINCRONA

All'interno di qualsiasi realizzazione logica digitale gli eventi possono essere temporizzati in modo sincrono, basandosi su un segnale di clock:

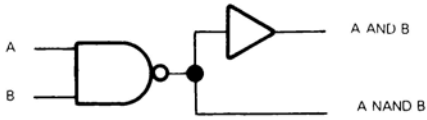


LOGICA ASINCRONA

ovvero in modo asincrono, basandosi su un segnale di uscita da un dispositivo che sta cambiando stato e così facendo scattare il cambiamento di stato di un altro dispositivo.



I gate singoli, comunque, sono dispositivi continui. Si consideri la seguente semplice sequenza logica.



TEMPO DI RITARDO DI GATE

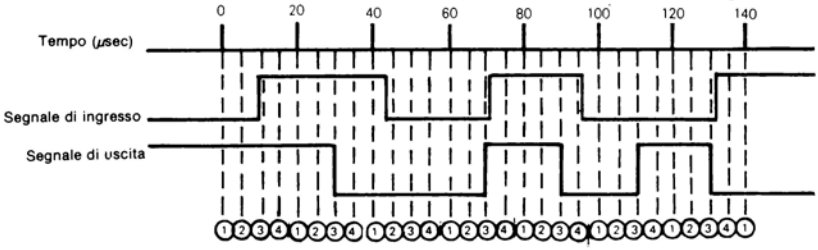
L'invertitore di segnale inverte continuamente il suo ingresso; un tempo di ritardo di gate di circa 10 nanosecondi è il solo ritardo tra i cambiamenti di stato dei segnali d'ingresso e uscita.

All'interno di un sistema a microcalcolatore, comunque, devono essere eseguite tre istruzioni prima che un segnale di uscita possa riflettere un cambiamento di stato del segnale d'ingresso.

Nell'improbabile eventualità che il sistema a microcalcolatore stia simulando un'invertitore e nient'altro, la sequenza di istruzione dell'invertitore potrebbe essere rieseguita continuamente come segue:

```
LOOP LDA   INVD   ; Carica l'Accumulatore da INVD
     CMA          ; Complementa l'Accumulatore
     STA   INVD   ; Immagazzina i contenuti dell'Accumulatore ad INVD
     JMP   LOOP   ; Riesegue la sequenza dell'invertitore di segnale
```

Dipendentemente dalla versione del tipo di microcalcolatore 8080 e dalla frequenza di clock, si impiegherà approssimativamente 20 microsecondi per eseguire una volta un ciclo di istruzione dell'invertitore di segnale, fornendo un periodo tra i cambiamenti di stato dell'ingresso che non è mai inferiore di 20 microsecondi ed il microcalcolatore realizza un invertitore di segnale che lavora sempre. Però **può esistere un ritardo maggiore di 20 microsecondi tra il cambiamento di stato del segnale d'ingresso ed il successivo segnale d'uscita corretto.** Questo può essere illustrato come segue:



① = LDA Esecuzione istruzione LDA
② = CMA Esecuzione istruzione CMA
③ = STA Esecuzione istruzione STA
④ = JMP Esecuzione istruzione JMP

Nella precedente illustrazione, le quattro istruzioni sono state mostrate dividendo ugualmente venti microsecondi, cosicchè ogni istruzione è stata eseguita in cinque microsecondi. In realtà non è così. Al Capitolo 6 vengono forniti i tempi di esecuzione di istruzioni, si vedrà che l'istruzione CMA, per esempio, richiede per essere eseguita un tempo considerevolmente inferiore rispetto a qualunque delle altre tre istruzioni. Per il momento si trascurerà questo dettaglio in modo da concentrare l'attenzione sul concetto secondo il quale si **deve fare molta attenzione alla sequenza di eventi all'interno del sistema a microcalcolatore.**

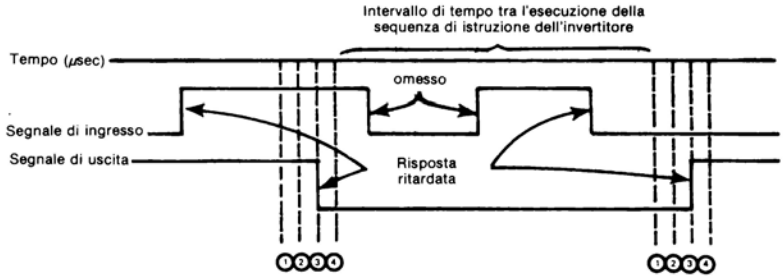
Indipendentemente da come e quando il "Segnale In" cambia stato, all'istante ① (quando l'istruzione LDA è eseguita) esso è lo stato di "Segnale In", che viene trasferito, come un digit binario, nel sistema a microcalcolatore.

L'inversione del digit binario effettivo avviene all'istante ②

Il digit binario invertito è convertito in "Segnale Out" all'istante ③ quando è eseguita l'istruzione STA.

Così il timing del "Segnale Out" può essere considerevolmente diverso dal timing del "Segnale In".

I problemi più complessi nascono quando la sequenza di istruzione dell'invertitore di segnale è appena una piccola parte di un grosso programma al microcalcolatore. In queste condizioni molti millisecondi possono trascorrere tra le esecuzioni ripetute della sequenza d'istruzione dell'invertitore. Se si lascia cambiare, l'inversione di segnale può essere completamente omessa. Nel migliore dei casi possono esserci considerevoli ritardi tra la variazione del segnale d'ingresso ed il conseguente segnale d'uscita. Questa situazione e illustrata come segue:



Inoltre ①, ②, ③ e ④ identificano rispettivamente l'esecuzione delle istruzioni LDA, CMA, STA e JMP.

Avendo sottolineato l'importanza del timing in un sistema a microcalcolatore, con le conseguenze dell'insufficienza di timing per il momento si sospenderà questo argomento Questo perchè **i problemi di timing svaniscono quando si simulano sequenze logiche intere rispetto ai singoli dispositivi.** Perciò le soluzioni ai problemi di timing dovrebbero essere guardate nel contesto dell'intera simulazione logica e si è ancora lontani da questo.

BUFFER, AMPLIFICATORI E CARICATORI DI SEGNALE

Osservata la temporizzazione si torna ora a considerare altri concetti fondamentali della logita digitale.

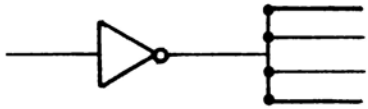
BUFFER **Un buffer di segnale aumenta il livello di corrente del segnale:**

Buffer

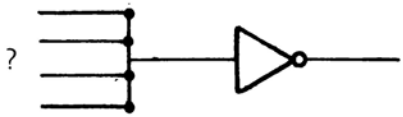
AMPLIFICATORE **Un driver amplificatore aumenta il livello di tensione del segnale:**

Amplificatore, driver

FAN OUT **Ogni dispositivo ha un ben definito fan out.** Il fan out definisce il numero di carichi paralleli che possono essere connessi ad un segnale d'uscita:



FAN IN Il dispositivi logici avranno anche un ben determinato fan in che indica il numero di carichi paralleli che possono essere connessi all'ingresso di un dispositivo:



Cosa succede di questi concetti quando la logica scompare in un programma di un microcalcolatore? La riposta è semplice: questi concetti scompaiono assieme alla logica digitale.

FAN IN
FAN OUT
CARICHI TTL
TAMPONAMENTO DEL SEGNALE

Ora ai pin effettivi del dispositivo microcalcolatore fisico il fan in ed il fan out rimangono dei concetti legittimi; i segnali che viaggiano tra i pin di singoli dispositivi a microcalcolatore possono richiedere di essere amplificati ed immagazzinati su buffers. Per esempio il fan out di un dispositivo CPU tipo 8080 può essere piccolo come uno o due carichi TTL (Transistor-Transistor Logic); questo significa che se più di uno o due dispositivi simili sono connessi ad un segnale di uscita il segnale d'uscita stesso avrà potenza insufficiente per trasmettere un segnale utile a tutti i dispositivi connessi. Perciò in tutte le semplici configurazioni a microcalcolatore le linee bus devono avere dei buffers.

CORRENTE DI LEAKAGE **Nella determinazione se le linee bus devono essere dotate di buffer non si dimentichi la corrente di leakage.** Per esempio se si hanno sedici dispositivi ROM connessi ad un sistema bus e deve essere selezionato un solo dispositivo (e perciò connesso) a qualsisi istante, non si assuma che il carico di segnale totale è dovuto alla ROM selezionata. I quindici dispositivi ROM non selezionati avranno una certa corrente di perdita; questo potrebbe risolversi nella necessità di un sistema tampone del bus.

All'interno di un programma di microcalcolatore, comunque, quando la logica è totalmente rappresentata da una sequenza d'istruzione del microcalcolatore, si sta trattando esclusivamente con digit binari — mai con livelli di tensione o corrente. Il fan in è infinito poichè lo stato di un digit binario può essere il risultato di un numero qualunque di calcoli logici. Il fan out è infinito poichè si può leggere lo stato di un digit binario spesso quanto si vuole. I buffers e gli amplificatori perdono significato poichè un digit binario non ha qualità equivalenti a tensione e corrente. Un digit binario offre una risoluzione infinita.
Si osserva un'altra cosa sull'invertitore di segnale simulato da un microcalcolatore.

Si considererà una fase concettuale più generale e si assumerà che l'invertitore di segnale sia contenuto all'interno di una sequenza logica, cosicchè nessun segnale d'ingresso o d'uscita è generato a qualunque pin del dispositivo microcalcolatore. In altre parole l'invertitore di segnale diventa una piccola parte di una funzione di trasferimento più grande.

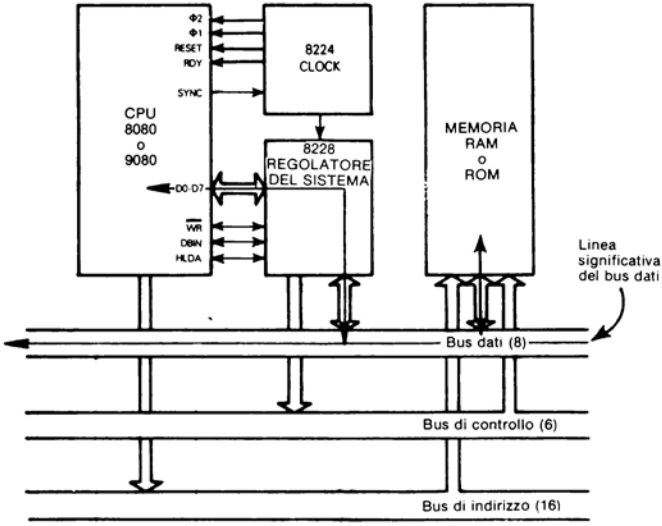
L'ingresso all'invertitore di segnale è un digit binario originato da qualche logica precedente.

L'uscita dell'invertitore di segnale è un altro digit binario che diventa l'ingresso alla logica successiva.

COMPLEMENTAZIONE DI UN BYTE DI MEMORIA

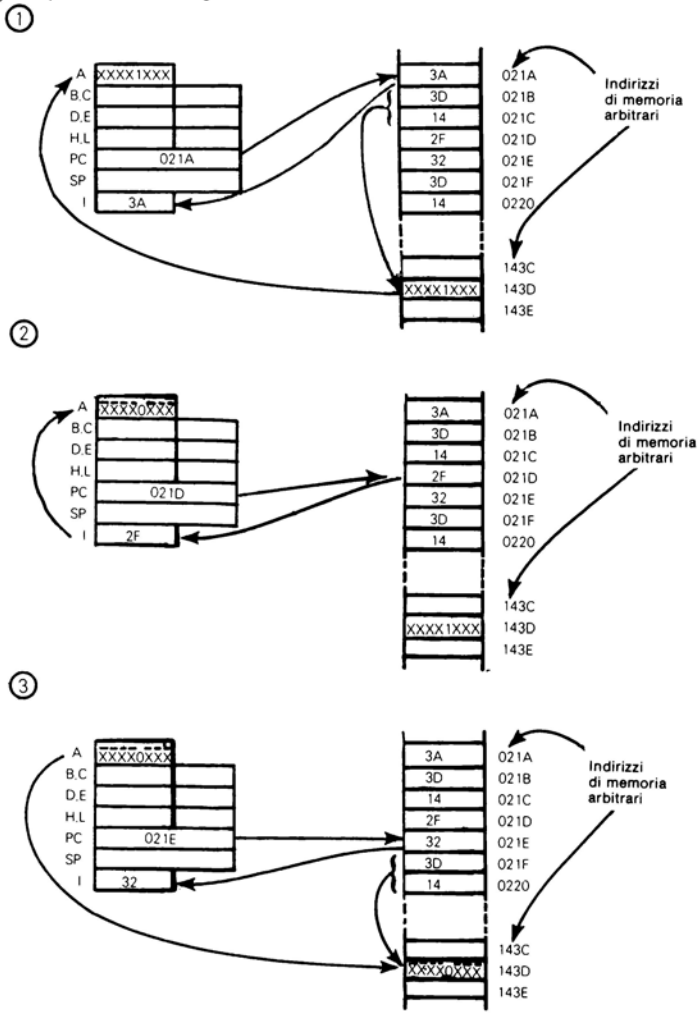
La logica esterna al sistema microcalcolatore non fornisce l'ingresso all'invertitore come un segnale che arriva al pin di un dispositivo a microcalcolatore e neppure il segnale invertito è trasmesso alla logica esterna attraverso un pin del dispositivo microcalcolatore. Invece è necessaria un'interfaccia tra la logica esterna ed il sistema microcalcolatore in alcuni punti significativi prima e dopo l'invertitore di segnale. **Gli invertitori di segnale possono quindi essere rappresentati da queste stesse tre istruzioni:**

```
LDA   INVD   ; Carica l'Accumulatore da INVD
CMA          ; Complementa
STA   INVD   ; Immagazzina i contenuti dell'Accumulatore ad INVD
```



La sorgente e destinazione diventano i bit della memoria dati: questo può essere illustrato come a pagina 2-15.

In termini di memoria e contenuti dei registri della CPU, la sequenza dell'invertitore di segnale procede come segue:



Con riferimento alla precedente illustrazione, le lettere A, B, C, D, E, H ed L identificano i sette registri di CPU del tipo 8080 — PC rappresenta il Contatore di Programma, SP il Puntatore dello Stack, I il registro d'istruzione.

I contenuti del byte della memoria dati $143D_{16}$ ed il registro A sono rappresentati in formato binario . X rappresenta qualsiasi digit binario. Si noti che è stato selezionato arbitrariamente il bit 3 come significativo.

Nella fase ① viene eseguita l'istruzione LDA. Questa istruzione impone il caricamento nell'Accumulatore dei contenuti del byte $143D_{16}$ della memoria dati.

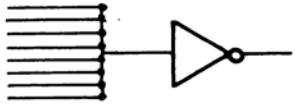
Durante la fase ② viene eseguita l'istruzione CMA. Questo fa sì che vengano complementati e contenuti dall'Accumulatore.

Durante la fase ③ i contenuti dell'Accumulatore sono ricaricati nel byte di memoria $143D_{16}$.

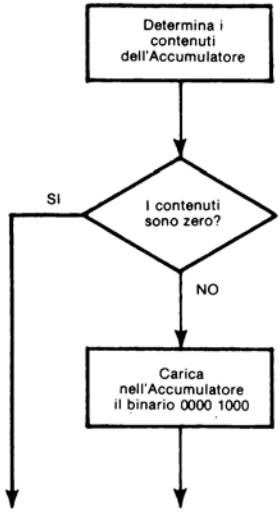
L'inversione di segnale è stata simulata invertendo i contenuti del bit 3 (assieme con ogni altro bit) del byte $143D_{16}$ della memoria dati.

FAN IN NEI PROGRAMMI DEL MICROCALCOLATORE

Da dove proviene l'ingresso dell'invertitore? **Si supponga, per chiarire un punto importante, che l'ingresso dell'invertitore sia l'OR di otto segnali.** Non si potrebbe realizzare l'OR cablato di questi otto segnali per creare un ingresso dell'invertitore come segue:



Ma, **presumendo che gli otto segnali siano rappresentati dai contenuti degli otto digit binari dell'Accumulatore, non si avrebbero problemi generando l'ingresso dell'invertitore attraverso la seguente sequenza logica:**

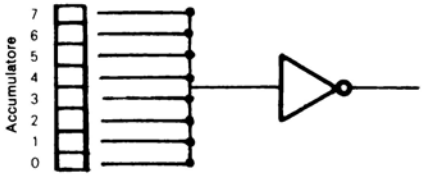


Il fan in logico è realizzato mediante questa sequenza di istruzione:

```
; Si assume che gli otto segnali sono dell'Accumulatore
  Ciascuno rappresentato da un bit dell'Accumulatore stesso
        ANA   A       ; Opera l'AND dell'Accumulatore con sé stesso per
                      ; porre ad 1 i flag di stato
        JZ    NEXT    ; Accumulatore uguale a 0 se il segnale d'ingresso è 0
        MVI   A,8     ; Accumulatore uguale ad 1 se il segnale d'ingresso è 1
NEXT    STA   INVD    ; Conserva l'ingresso dell'invertitore
```

La precedente sequenza d'istruzione è una realizzazione, attraverso il programma del microcalcolatore, dell'OR cablato di otto segnali. Si esamini come opera la logica di istruzione.

Si assumerà che gli otto segnali d'ingresso siano inizialmente reppresentati dallo stato degli otto digit binari dell'Accumulatore:



Si assumerà inoltre che, in accordo con l'illustrazione precedente, il bit 3 del byte dati sarà il bit di segnale significativo per l'invertitore.

Poichè l'ingresso dell'invertitore è l'OR cablato di otto segnali, la logica del programma deve porre il bit 3 dell'Accumulatore ad 1 se qualunque bit dell'Accumulatore è diverso da zero; invece il bit 3 dell'Accumulatore deve essere posto a 0 se tutti i bit dell'Accumulatore stesso sono 0. I contenuti dell'Accumulatore sono poi immagazzinati nel byte di memoria dati rappresentato dalla label INVD. Con riferimento alle precedenti illustrazioni INVD potrebbe essere una label rappresentante il byte di memoria $143D_{16}$.

Questo è il modo di operare della sequenza di quattro istruzioni precedentemente illustrata.

DETERMINAZIONE DELLO STATO OPERANDO L'AND DI UN REGISTRO CON SE' STESSO

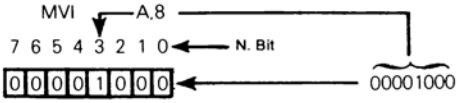
Naturalmente non si conoscono i contenuti iniziali dell'Accumulatore, così occorre determinarli ponendo opportunamente i flag di stato della CPU. Per fare questo si opera l'AND dei contenuti dell'Accumulatore con sé stessi. Questa operazione non cambia i contenuti dell'Accumulatore ma i flag di stato vengono assegnati. Si è interessati al solo stato Zero che sarà posto ad 1 se l'AND genera un risultato zero; il flag di stato Zero sarà posto a zero in caso contrario.

D'altra parte l'AND dell'Accumulatore con sé stesso sarà zero solo se l'Accumulatore contiene zero:

```
0 0 0 0 0 0 0 0
• • • • • • • •
0 0 0 0 0 0 0 0
" " " " " " " "
0 0 0 0 0 0 0 0
```

Così dopo l'esecuzione dell'istruzione ANA, se lo stato Zero è 1, allora il bit 3 dell'Accumulatore deve già essere 0, che è quanto si vuole. Nessuna operazione è richiesta e si salta all'istruzione STA.

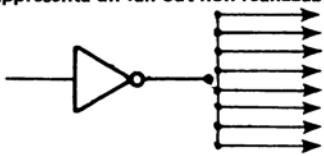
Se il bit Zero era 0 allora uno o più bit dell'Accumulatore sono diversi da zero. La istruzione MVI carica un 1 nel bit 3 dell'Accumulatore:



Infine viene eseguita l'istruzione STA per caricare il segnale d'ingresso dell'invertitore nel byte appropriato della memoria dati.

Si supponga ora che l'uscita dell'invertitore sia distribuita a numerosi dispositivi successivi.

La logica seguente rappresenta un fan out non realizzabile:



FAN OUT NEI PROGRAMMI DI MICROCALCOLATORE

All'interno di un programma di microcalcolatore l'intero concetto di fan out scompare. L'uscita dell'invertitore può essere accessibile un numero indefinito di volte mediante una semplice riesecuzione di un'istruzione LDA:

```
LDA     INVD     ; Carica l'uscita dell'invertitore nell'Accumulatore
-
-
-
LDA     INVD     ; Carica l'uscita dell'invertitore nell'Accumulatore
-
-
-
LDA     INVD     ; Carica l'uscita dell'invertitore nell'Accumulatore
-
-
-
LDA     INVD     ; Carica l'uscita dell'invertitore nell'Accumulatore
-
-
-
LDA     INVD     ; Carica l'uscita dell'invertitore nell'Accumulatore
```

E per quanto riguarda gli amplificatori ed i buffers? Chiaramente nel contesto dei dati binari immagazzinati nella memoria, essi non hanno alcun significato. Se gli amplificatori ed i buffers sono presenti a causa delle caratteristiche elettriche dei chip del processore e della memoria ciò non ha nulla in relazione con la funzione logica realizzata da un programma al microcalcolatore.

SIMULAZIONE AL MICROCALCOLATORE DEGLI INVERTITORI SESTUPLI 7404/05/06/07

Questi quattro invertitori sestupli differiscono solo nelle loro caratteristiche elettriche:

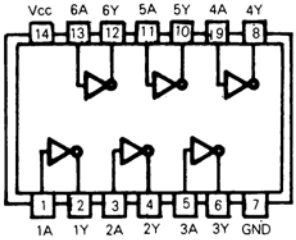
Il 7404 è un invertitore sestuplo semplice.

Il 7405 è un invertitore sestuplo con uscite a collettore aperto.

Il 7406 è un invertitore sestuplo buffer/driver a collettore aperto, uscite ad alta tensione.

Poichè questi tre dispositivi differiscono solo nelle loro caratteristiche elettriche, all'interno di una simulazione in linguaggio assembly al microcalcolatore essi sono identici. Si osservi il 7404. Esso consiste di sei invertitori di segnale indipendenti

che possono essere illustrati come segue:



$Y = \overline{A}$

La sequenza d'istruzione per rappresentare un invertitore sestuplo è identica alle tre istruzioni della sequenza di istruzione dell'invertitore di segnale singolo, poichè i microcalcolatori tipo 8080 sono dispositivi ad otto bit paralleli. Così questa sequenza di istruzione dell'invertitore inverte otto digit rappresentati come segue all'interno di una sequenza di istruzione del microcalcolatore:

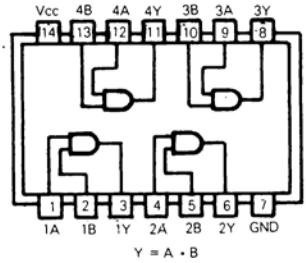
```
LDA   INVD   ; Carica l'Accumulatore da INVD
CMA          ; Complementa
STA   INVD   ; Immagazzina i contenuti dell'Accumulatore ad INVD
```

Si identificheranno arbitrariamente i bit significativi implicati dall'invertitore sestuplo, come segue:

Si noti che la precedente selezione di bit significativi è completamente arbitraria. Non c'è assolutamente nessuna ragione pratica o teorica che favorisce qualsiasi assegnazione di bit rispetto a qualsiasi altra.

SIMULAZIONE AL MICROCALCOLATORE DEI 7408/09 GATES AND QUADRUPLI A DUE INGRESSI POSITIVI

Questi due dispositivi forniscono quattro gates AND indipendenti a due ingressi ed una uscita che possono essere illustrati come segue:

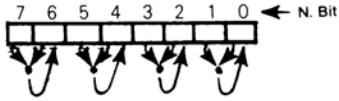


Il 7409 ha uscite a collettore aperto che lo differenzia dal 7408. Questa differenza non ha significato nella simulazione di programma al microcalcolatore; perciò i due dispositivi da questo punto di vista possono essere considerati identici.

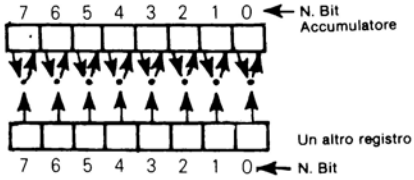
DUE FUNZIONI D'INGRESSO

Dal punto di vista dei programmi del microcalcolatore, la differenza più significativa tra un gate AND 7408 ed un invertitore 7404 non è la funzione logica ma il fatto che il 7408 è un dispositivo a due ingressi. Concettualmente si potrebbe immaginare una simulazione del 7404 nelle due maniere seguenti:

1) Gli otto segnali d'ingresso sono caricati nel registro Accumulatore della CPU. Ogni bit pari subisce l'AND col bit alla sua destra.
   Il risultato è depositato nei bit pari di ogni coppia:



2) I due set di quattro ingressi sono caricati nell'Accumulatore ed il risultato da un'altro registro viene riportato nell'Accumulatore.



Esaminando il set d'istruzione del microcalcolatore 8080 si troverà che il secondo metodo di simulazione del 7408 è quello naturale. Questa è la sequenza d'istruzione richiesta:

```
LDA   SRCA   ; Carica il primo set d'ingressi da SRCA
MOV   B,A    ; Conserva nel registro B
LDA   SRCB   ; Carica il secondo set d'ingressi da SRCB
ANA   B      ; AND di B con A
STA   DST    ; Conserva il risultato in DST
```

ASSEGNAZIONE DI LABEL DEL PROGRAMMA SORGENTE

Se l'impiego delle label SRCA, SRCB e DST genera confusione di seguito vengono chiarite. Si avrà a disposizione una quantità di memoria che può variare da 256 byte a 65.536 byte. Ciascuna delle labels SRCA, SRCB e DST identifica un byte di memoria. Nell'istante in cui scrive il programma sorgente è importante l'esatta identificazione di ogni byte di memoria mediante label. Quando eventualmente si assembla il programma la lista dell'assemblatore stamperà una mappa di memoria. La mappa di memoria identificherà lo esatto byte di memoria associato con ogni label che si è impiegato. Esaminando la mappa di memoria si sarà in grado di determinare se tutte le assegnazioni di label sono valide oppure no. Se tutti gli assegnamenti di label non sono validi occorre prendere un provvedimento adeguato. Questo consiste nell'aggiungere più memoria alla configurazione del microcalcolatore oppure si dovrà riscrivere il programma sorgente in modo da impiegare più razionalmente la memoria disponibile.

Il problema delle label e locazioni di memoria a questo livello di discussione è irrilevante. Si immagini semplicemente che ogni label indirizzi un byte di memoria specifico. Non ci si preoccupi se il byte di memoria eventualmente sarà indirizzato ed il problema scomparirà.

La sequenza di istruzione per la simulazione del 7408 precedentemente illustrato non è il solo modo in cui può essere simulato un 7408.

Prima si considerino alcune variazioni minori. I registri C, D, E, H ed L della CPU potrebbero essere impiegati al posto del registro B per contenere il secondo ingresso dati. Questo è un esempio:

```
LDA   SRCA   ; Carica il primo set di ingressi da SRCA
MOV   C,A    ; Conserva nel registro C
LDA   SRCB   ; Carica il secondo set di ingressi da SRCB
ANA   C      ; Opera l'AND di C con A
STA   DST    ; Conserva il risultato in DST
```

IMPIEGO DELL'INDIRIZZAMENTO DI MEMORIA IMPLICATO

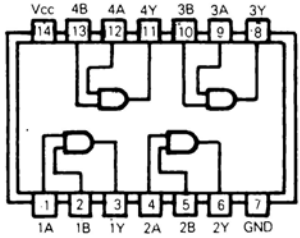
L'impiego dei registri H ed L per contenere il secondo ingresso non è incoraggiante. L'impiego primario di questi due registri è di conservare un indirizzo della memoria dati.

Per esempio le istruzioni LDA e STA potrebbero essere sostituite come segue:

```
LXI   H,SCRA   ; Carica l'indirizzo del primo set d'ingressi in H, L
MOV   A,M      ; Carica il primo set d'ingressi in A
LXI   H,SCRB   ; Carica l'indirizzo del secondo set di ingressi in H, L
ANA   M        ; Opera l'AND del secondo set d'ingressi con A
LXI   H,DST    ; Carica l'indirizzo di destinazione in H, L
MOV   M,A      ; Immagazzina il risultato in DST
```

LA SIMULAZIONE AL MICROCALCOLATORE DI UN 7411 TRIPLO, TRE-INGRESSI, GATE AND POSITIVO

La principale differenza tra il gate AND 7411 ed il 7408 è il numero di segnali d'ingresso. Il 7411 genera tre segnali d'uscita, ognuno dei quali è l'AND di tre ingressi:



Y = A • B

TRE FUNZIONI D'INGRESSO

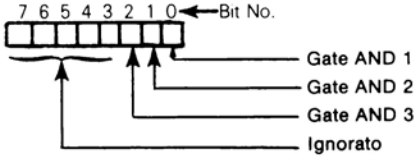
Anche qui occorre fare delle scelte. Si può caricare i tre set d'ingressi in tre registri CPU (l'Accumulatore e due altri registri), quindi eseguire due AND prima di ri-immagazzinare il risultato:

```
UNO    LDA   SRCA   ; Carica il primo set d'ingressi da SCRA
DUE    MOV   B,A    ; Conserva nel registro B
TRE    LDA   SRCB   ; Carica il secondo set d'ingressi da SCRB
QUAT.  MOV   C,A    ; Conserva nel registro C
CINQ.  LDA   SRCC   ; Carica il terzo set d'ingressi da SCRC
SEI    ANA   B      ; AND di B con A
SETTE  ANA   C      ; AND di C con A
OTTO   STA   DST    ; Conserva il risultato in DST
```

Le istruzioni nella precedente sequenza sono state contrassegnate con labels così da

rendere la descrizione seguente più facile da capire. Le istruzioni non necessitano di label allo scopo di soddisfare le richieste di un programma sorgente in linguaggio assembly.

Quando viene eseguita l'istruzione UNO, un valore ad 8-bit è caricato nell'Accumulatore dal byte di memoria indirizzato dalla label SCRA. Si assumerà che gli ingressi del gate AND siano rappresentati come segue:



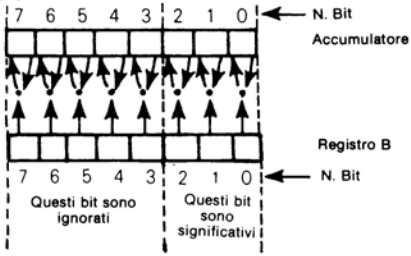
Si comprende che l'assegnazione dei bit dati sopra illustrata è completamente arbitraria. E' necessario che tutti gli ingressi successivi siano consistenti.

Dopo che è stata eseguita l'istruzione UNO il primo set di ingressi si trova nell'Accumulatore. L'Accumulatore è il solo registro della CPU in cui i dati possono essere caricati se si impiega l'indirizzamento diretto. Il primo degli ingressi deve perciò essere conservato in un altro registro cosicchè l'Accumulatore è libero per accettare un secondo set d'ingressi. L'istruzione DUE muove i contenuti dell'Accumulatore al registro B.

Le istruzioni TRE e QUATTRO caricano il secondo set d'ingressi nell'Accumulatore, quindi lo muovono al registro C. Si assume che l'assegnazione di bit di questo secondo set di ingressi è identica all'assegnazione precedentemente illustrata per il primo ingresso.

Il terzo ed ultimo set d'ingressi viene caricato nell'Accumulatore dall'istruzione CINQUE.

L'istruzione ANA opera l'AND dei contenuti del registro della CPU con i contenuti dell'Accumulatore, lasciando il risultato nell'Accumulatore. L'istruzione SEI esegue il primo AND come segue:



L'istruzione SETTE esegue la seconda operazione di AND. Questa volta l'AND ricorre tra l'Accumulatore ed il Registro C. L'Accumulatore inizialmente conserva il risultato dell'AND con B, precedentemente illustrato. Dopo che è stata eseguita l'istruzione SETTE l'AND dei tre ingressi si trova nell'Accumulatore.

L'istruzione OTTO riporta il risultato finale nel byte di memoria indirizzato dalla label DST. Il gate AND 7411 è stato così completamente simulato.

Si consideri ora una simulazione alternativa dei gates AND 7411. Si può caricare il primo ingresso nell'Accumulatore ed il secondo in un altro registro. Dopo l'operazione di AND di questi due ingressi è possibile caricare il terzo ingresso nello stesso "altro" registro,operare l'AND di quest'ultimo con il risultato del primo AND e quindi riportare il risultato:

```
UNO     LDA     SRCA    ; Carica il primo set d'ingressi da SCRA
DUE     MOV     B,A     ; Conserva nel registro B
TRE     LDA     SRCB    ; Carica il secondo set di ingressi da SCRB
QUAT.   ANA     B       ; AND di B con A, il risultato è in A
CINQ.   MOV     B,A     ; Conserva il risultato in B
```

```
SEI     LDA     SRCC     ; Carica il terzo set d'ingressi da SCRC
SETT    ANA     B        ; AND di B con A
OTTO    STA     DST      ; Conserva il risultato in DST
```

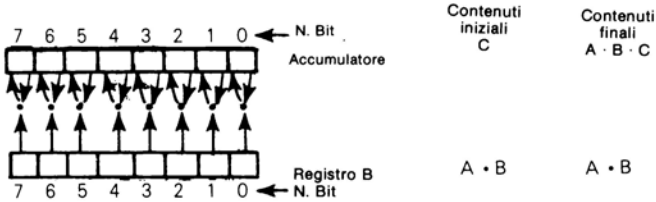
Si confronti questa seconda simulazione del GATE AND 7411 con la prima simulazione. Le istruzioni UNO, DUE e TRE sono identiche alla prima simulazione, un set d'ingressi è nel Registro B ed un secondo set d'ingressi è nell'Accumulatore.
La situazione è questa:

Gli ingressi A sono nell'Accumulatore
Gli ingressi B sono nel Registro B

Ora invece di portare il terzo set di ingressi immediatamente in un registro CPU, si esegue l'istruzione QUAT, che genera l'AND dei primi due ingressi. Poichè questo AND è generato nell'Accumulatore si conserva il risultato nel registro B l'istruzione CINQ. Quindi l'effetto netto è:

A • B nel Registro B

Ora l'istruzione SEI carica il terzo set d'ingressi nell'Accumulatore. L'istruzione SETTE opera l'AND del terzo set di ingressi con il risultato del primo AND come segue:



L'istruzione OTTO conserva il risultato dall'Accumulatore nel byte di memoria indirizzato dalla label DST.

MINIMIZZAZIONE DEGLI ACCESSI AI REGISTRI CPU

Qual'è la migliore simulazione dei gates 7411 AND? Chiaramente la seconda scelta. Esiste un problema non ovvio associato con l'impiego indiscriminato dei registri della CPU. E' stato deciso arbitrariamente che il Registro B conterrà il secondo ingresso. Così mentre si sta simulando i gates AND 7411, senza considerare cosa precede o segue, la selezione del Registro B è arbitraria; la sua selezione è fatta senza pensare alle conseguenze.

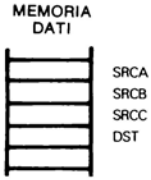
CONFLITTI NELL'UTILIZZAZIONE DEI REGISTRI DELLA CPU

Invariabilmente, una sequenza d'istruzione come la simulazione dei gates AND 7411 è appena una piccola parte di un grosso compito. Ora occorre preoccuparsi se l'impiego del Registro B come sede del secondo ingresso interferirà con l'impiego precedente o successivo del Registro B stesso. Un errore di programmazione molto comune coinvolge i conflitti di utilizzazione dei registri della CPU. Per esempio cosa succede se qualche fase logica precedente utilizza il registro B per conservare un valore dati intermedio? Ora la simulazione del 7411 distruggerà i dati che sono stati temporaneamente immagazzinati in questo registro.

Allo scopo di ridurre i conflitti dei registri della CPU è sempre preferibile scegliere una sequenza d'istruzione che impieghi meno registri CPU possibile, senza introdurre penalizzazioni significative. In questo caso non c'è penalizzazione significativa nella simulazione dei gates AND 7411 impiegando il solo Registro B anzichè B e C. Impiegando solo il registro B è perciò il metodo migliore.

INDIRIZZAMENTO IMPLICATO

Si considera ora la simulazione dei gates AND 7411 impiegando l'indirizzamento implicato. Si assume che i tre ingressi ai gates AND siano immagazzinati in bytes sequenziali della memoria dati e la destinazione segua l'ultimo byte sorgente, nel modo indicato:



Ora impiegando l'indirizzamento implicato si ha la seguente sequenza d'istruzioni:

```
UNO     LXI     H,SRCA    ; Carica il primo indirizzo sorgente in H, L
DUE     MOV     A,M       ; Carica la seconda sorgente nell'Accumulatore
TRE     INX     H         ; Incrementa l'indirizzo implicato
QUAT.   ANA     M         ; AND dell'Accumulatore con la seconda sor-
                          ; gente
CINQ.   INX     H         ; Incrementa l'indirizzo implicato
SEI     ANA     M         ; AND dell'Accumulatore con la terza sorgente
SETTE   INX     H         ; Incrementa l'indirizzo implicato
OTTO    MOV     M,A       ; Conserva il risultato
```

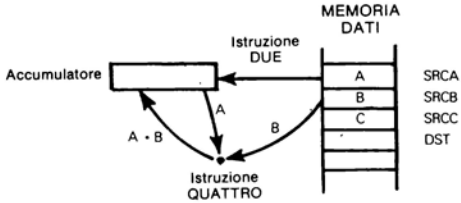
La sequenza d'istruzione sarà eseguita come segue:

L'istruzione UNO carica l'indirizzo del primo byte sorgente nei registri H ed L.

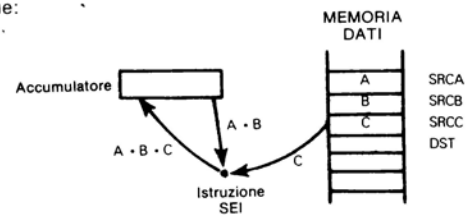
L'istruzione DUE muove i contenuti del byte di memoria indirizzato dai registri H ed L nell'Accumulatore.

L'istruzione TRE incrementa l'indirizzo a 16 bit dei registri H ed L cosicchè essi ora indirizzano SRCB.

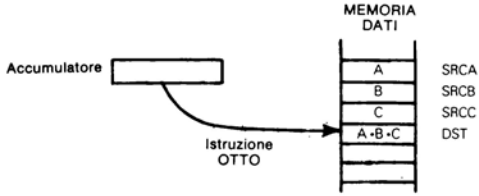
L'istruzione QUAT. opera l'AND dei contenuti dell'Accumulatore con la seconda sorgente, indirizzata dai registri M ed L. Il risultato è conservato nell'Accumulatore. Questo può essere illustrato come segue:



Le istruzioni CINQ.e SEI incrementano l'indirizzo implicato e ripetono l'operazione AND, questa volta tra il terzo ingresso con l'AND dei primi due. Questo può essere illustrato come segue:



L'indirizzo in H ed L viene ancora incrementato per puntare a DST. L'istruzione OTTO conserva il risultato nella destinazione come segue:



CONFRONTO DELL'IMPIEGO DI MEMORIA E DELLA VELOCITA' DI ESECUZIONE

Si hanno ora questi tre programmi, ciascuno simula i gates AND 7411:

Il Programma 1 impiega l'indirizzamento diretto e tre registri CPU.

Il Programma 2 impiega l'indirizzamento diretto e due registri CPU.

Il Programma 3 impiega l'indirizzamento implicato.

Si confronti il numero dei bytes del programma oggetto richiesti per immagazzinare ogni programma ed il numero di cicli di clock della CPU richiesti per eseguire ogni programma. I risultati sono riassunti nella Tabella 2-1. La Tabella 2-1 comprende i mnemonici di istruzione per ogni programma per aiutare a seguire come vengono calcolati i bytes totali del programma oggetto ed i cicli di esecuzione. Si veda il Capitolo 6 per i dati necessari alla verifica della Tabella 2-1.

Tabella 2-1. Confronto tra impiego di memoria e velocità di esecuzione del programma per la simulazione dei 7411 AND gates.

PROGRAMMA 1			PROGRAMMA 2			PROGRAMMA 3		
MNEMONICO	BYTE	CICLI	MNEMONICO	BYTE	CICLI	MNEMONICO	BYTE	CICLI
LDA	3	13	LDA	3	13	LXI	3	10
MOV[1]	1	5	MOV[1]	1	5	MOV[2]	1	7
LDA	3	13	LDA	3	13	INX	1	5
MOV[1]	1	5	ANA[1]	1	5	ANA[2]	1	7
LDA	3	13	MOV[1]	1	13	INX	1	5
ANA[1]	1	4	LDA	3	4	ANA[2]	1	7
ANA	1	4	ANA[1]	1	4	INX	1	5
STA	3	13	STA	3	13	MOV[2]	1	7
TOTALE	16	70	TOTALE	16	70	TOTALE	10	53

[1] Versione registro-registro dell'istruzione
[2] Versione registro-memoria dell'istruzione

CONFRONTO INDIRIZZAMENTO DIRETTO ED IMPLICATO

I Programmi 1 e 2 hanno un'utilizzazione di memoria e velocità di esecuzione identiche — cosa non sorprendente poichè essi variano la sequenza in cui sono eseguite le stesse istruzioni. **Il Programma 3 adotta una filosofia completamente diversa per la simulazione dei gates AND 7411 mediante l'impiego dell'indirizzamento di memoria implicato invece di quello diretto. In questo modo si risparmiano sei bytes di memoria ed il programma viene eseguito nel 76% del tempo.** Però il Programma 3 pone una restrizione addizionale alla simulazione: le tre sorgenti dati e la destinazione devono occupare quattro bytes contigui della memoria dati.

CLASSIFICA DELLE VARIAZIONI DEL PROGRAMMA

Come si possono classificare queste tre scelte di simulazione?

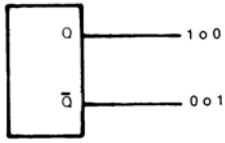
SI è già concluso che il Programma 2 batte il Programma 1, perchè il Programma 1 fa un impiego gratuito di un ulteriore registro CPU. Il programma 3 è chiaramente migliore del 2 purchè sia tollerabile la restrizione sulle locazioni della sorgente dei dati di destinazione.

SIMULAZIONE AL MICROCALCOLATORE DI UN FLIP-FLOP 7474 DUALE, TIPO TRIGGER A GRADINO POSITIVO CON PRESET E CLEAR

Prima di considerare in particolare il flip-flop 7474, si considerano i flip-flop in generale. Prima alcune definizioni.

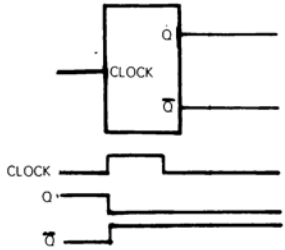
UNA DESCRIZIONE LOGICA DIGITALE DEI FLIP-FLOP

Un flip-flop è un dispositivo logico bistabile che può permanere in una di due condizioni stabili. I flip-flop del tipo 7474 hanno due uscite Q e $\overline{Q}$ cosicchè le due condizioni stabili possono essere rappresentate come segue:



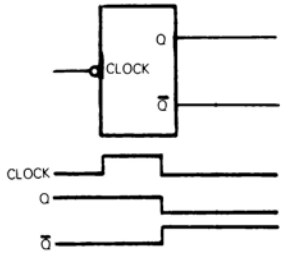
TRIGGER A GRADINO POSITIVO

Un segnale di clock origina il cambiamento del flip-flop da una condizione bistabile all'altra.



TRIGGER A GRADINO NEGATIVO

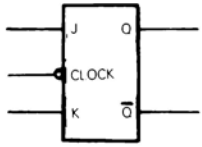
Un flip-flop con trigger a gradino negativo cambia stato quando avverte una transizione del segnale di clock da uno a zero:



FLIP-FLOP JK

Il flip-flop JK precondiziona le uscite Q e $\overline{Q}$ che saranno generate mediante il trigger dei successivi gradini di clock, come segue:

Stato di J e K in funzione del segnale di clock		Uscite generate in funzione del segnale di clock	
J	K	Q	$\overline{Q}$
1	0	1	0
0	1	0	1
0	0	Permane nello stato in cui si trova. Cambia stato indipendentemente da quello precedente	
1	1		



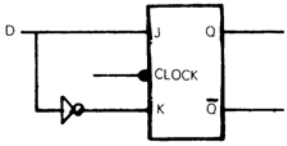
SEGNALE DI CLOCK

Nella tabella precedente, il "segnale di clock" sarà una transizione da zero ad uno per un dispositivo con trigger a gradino positivo, sarà la transizione da uno a zero per un dispositivo con trigger a gradino negativo. Questa definizione di "segnale di clock" si applica anche al flip-flop tipo D descritto di seguito.

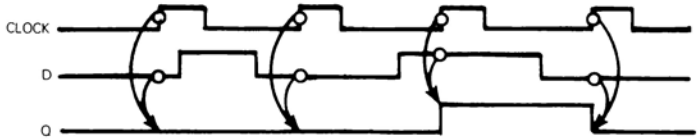
FLIP-FLOP TIPO D

Invertendo l'ingresso J per generare l'ingresso K si ottiene il flip-flop tipo D. Le caratteristiche del flip-flop D sono le seguenti:

Stato di J e K in funzione del segnale di clock		Uscite generate in funzione del segnale di clock	
J = D	K = $\overline{J}$	Q	$\overline{Q}$
1	0	1	0
0	1	0	1



Qui c'è un trigger a gradino positivo ed il diagramma di timing del flip-flop D risulta:



Un flip-flop D avrà quindi in uscita le condizioni d'ingresso che esistono all'impulso di clock precedente.

PRESET DEL FLIP-FLOP
CLEAR DEL FLIP-FLOP

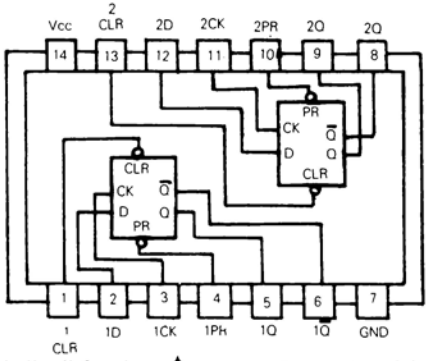
La presenza di un ingresso Preset significa che il flip-flop può essere forzato all'uscita Q=1 e $\overline{Q}$=0. Il Preset attivo forza questa condizione.

Un ingresso Clear è l'opposto dell'ingresso Preset. Quando è attivo l'ingresso Clear forza Q=0 e $\overline{Q}$=1.

Combinando le definizioni appena fornite, questo è quanto si ha da un flip-flip tipo 7474:

TABELLA DI FUNZIONE

INGRESSI				USCITE	
1PR o 2PR	1CLR o 2CLR	1CK o 2CK	1D o 2D	1Q o 2Q	1$\overline{Q}$ o 2$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

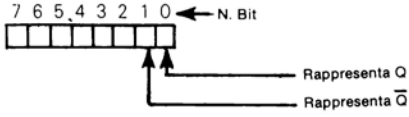


Nella precedente tabella di funzione ↑ rappresenta una transizione del clock da zero a uno, H* significa uno stato instabile, Q_0 è lo stato precedente di Q. X significa "non importa".

UNA SIMULAZIONE IN LINGUAGGIO ASSEMBLY DEI FLIP-FLOP

Ora il primo problema, cercando di simulare un flip-flop 7474, è il fatto che non c'è segnale di clock all'interno del set d'istruzione del microcalcolatore. Invece si deve assumere che gli eventi sono soggetti al trigger mediante la esecuzione di un opportuna istruzione piuttosto che una transizione del segnale di clock.

Come si rappresenteranno le uscite Q e $\overline{Q}$? Due bit di memoria potrebbero essere impiegati per rappresentare queste due uscite:

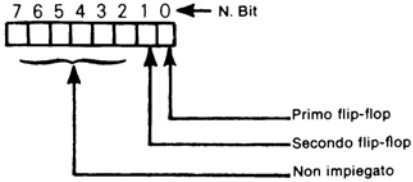


Poichè si sta trattando con dati, anzichè segnali, $\overline{Q}$ è ridondante. Perciò il singolo flip-flop fa capo ad un singolo bit di memoria. Un dispositivo 7474, contenendo due flip-flop, fa capo a due bit di memoria, uno per ogni flip-flop realizzato sul chip.

Non c'è nulla di sorprendente in questa conclusione. Ogni bit della memoria di lettura/scrittura del microcalcolatore è un elemento bistabile semplice; esso potrebbe, in realtà, essere un flip-flop.

La logica di un flip-flop 7474 può essere rappresentato dalle istruzioni che azzerano un bit di memoria, pongono il bit di memoria ad 1 oppure immagazzinano un digit binario non noto nel bit di memoria.

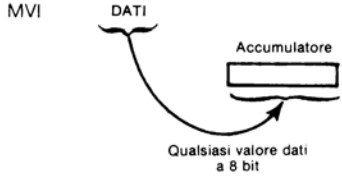
Si supponga che i bit di memoria siano assegnati come segue:



La tabella di funzione del 7474 ora assume queste istruzioni:

	Preset	Clear	D	Primo flip-flop	Secondo flip-flop
	L	H	X	MVI 1	MVI 2
o	H	H	H	STA FLP	STA FLP
	H	L	X	MVI 0	MVI 0
o	H	H	L	STA FLP	STA FLP
	L	L	X	Non applica	

Rispetto alla tabella precedente, l'istruzione MVI agisce come segue sui contenuti dell'Accumulatore:



L'istruzione STA immagazzina i contenuti risultanti dell'Accumulatore in una parola identificata dalla label FLP. I bit 0 ed 1 della parola di memoria identificata dalla label FLP sono presunti equivalenti a 2 flip-flop del dispositivo 7474.

SIMULAZIONE AL MICROCALCOLATORE DEL FLIP-FLOP IN GENERALE

In conclusione un flip-flop diventa un singolo bit della memoria di lettura/scrittura all'interno di un sistema a microcalcolatore.

All'interno di un sistema microcalcolatore tutti i flip-flop sono uguali. La logica del flip-flop si riduce a queste quattro domande:

1) Quando si deve eseguire un'istruzione per porre un bit di memoria ad 1?

2) Quando si deve eseguire un'istruzione per ripristinare un bit di memoria a 0?

3) Quando si deve eseguire un'istruzione per immagazzinare un digit binario in un bit di memoria?

4) Quando si deve eseguire un'istruzione per leggere i contenuti di un bit di memoria?

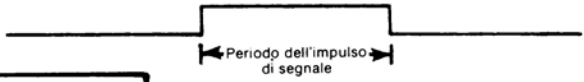
LA SIMULAZIONE AL MICROCALCOLATORE DEI DISPOSITIVI IN TEMPO REALE

Ci sono due tipi di dispositivi in tempo reale che saranno considerati: quello one-shot (comprendente i multivibratori monostabili) ed il flip-flop master-slave. Più specificatamente saranno descritti i dispositivi:

- Il multivibratore monostabile Signetics 555
- Il multivibratore monostabile 74121
- Il flip-flop master-slave J-K duale 74107 con Clear

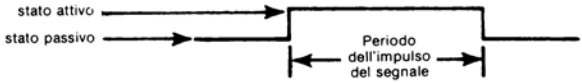
ONE-SHOT

Uno one-shot è un dispositivo che genera un impulso di segnale con un periodo di tempo specifico:



MULTIVIBRATORE MONOSTABILE

Un multivibratore monostabile è un dispositivo con uno stato stabile o passivo. Esso produce un segnale d'uscita one-shot, come sopra illustrato, dove l'impulso è in corrispondenza dello stato instabile o attivo:



Il dispositivo è un "multivibratore" perchè può far uscire una sequenza di segnali — quanti sono i segnali di clock. In altre parole l'uscita di un multivibratore consiste di una successione di segnali one-shot.

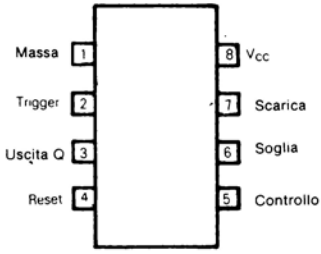
Il periodo di tempo dell'impulso segnale è un valore in tempo reale — esso è un numero finito di microsecondi, o millisecondi, oppure anche secondi.

FLIP-FLOP MASTER-SLAVE

Il flip-flop master-slave è un flip-flop che genera segnali di uscita basati sulla condizione dei segnali d'ingresso di un certo tempo precedente. Anche qui si incontra un valore in tempo reale — il ritardo tra ingressi ed uscite.

IL MULTIVIBRATORE MONOSTABILE 555

Il multivibratore Signetic 555 può essere illustrato come segue:



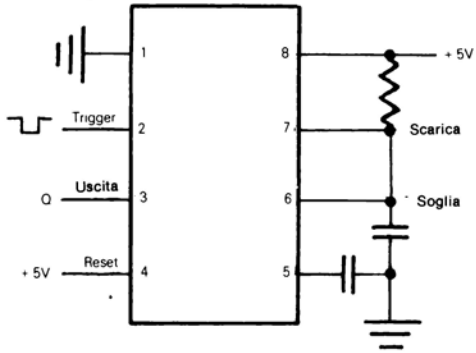
Il gradino negativo del segnale di clock all'ingresso Trigger (pin 2) origina una transizione da negativo a positivo dell'Uscita Q. La durata dell'uscita ad alto livello a Q è controllata da un circuito resistenza/condensatore connesso ai pin Scarica e Soglia (7 e 6 rispettivamente).

Reset è un'ingresso di reset standard; un ingresso basso manterrà l'uscita Q bassa.

Il pin Controllo è impiegato per controllare la tensione all'interno del multivibratore; esso non è significativo per una comprensione globale di come opera il dispositivo 555.

I pin di massa ed alimentazione (1 ed 8 rispettivamente) non necessitano di chiarimenti.

Questo è un modo in cui può essere configurato il multivibratore monostabile 555:

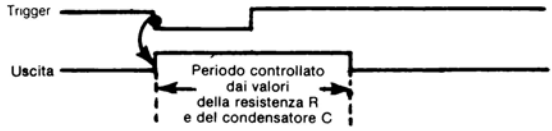


Non appena viene avvertito il livello di segnale da alto a basso all'ingresso trigger, il condensatore tra il pin 6 e massa si carica. I livelli di segnale ai pin di soglia e scarica, attraverso la resistenza R ed il condensatore C, controllano il periodo durante il quale l'uscita Q sarà alta. Questo periodo di tempo è dato dalla seguente equazione:

$$T = 1 \cdot 1RC$$

Dove T è il tempo in secondi
R è la resistenza in Megaohm
C è la capacità in microfarad

Un impulso del segnale d'uscita è generato come segue:

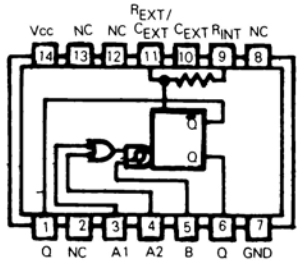


IL MULTIVIBRATORE MONOSTABILE 74121

Il multivibratore monostabile 74121 può essere illustrato come segue:

TABELLA DI FUNZIONE

INGRESSI			USCITE		
A1	A2	B	Q	$\overline{Q}$	
L	X	H	L	H	Uscite monostabili
X	L	H	L	H	
X	X	L	L	H	
H	H	X	L	H	
H	↓	H	⎍	⎎	Uscite One-shot
↓	H	H	⎍	⎎	
↓	↓	H	⎍	⎎	
L	X	↑	⎍	⎎	
X	L	↑	⎍	⎎	



Un ingresso basso costante ad A1, A2 o B manterrà il multivibratore monostabile nella sua condizione stabile con una uscita Q bassa ed una uscita $\overline{Q}$ alta. Ingressi alti ad A1 ed A2 hanno lo stesso effetto.

Ci sono cinque combinazioni di segnali d'ingresso che genereranno uscite one-shot. Queste combinazioni di segnali di ingresso sono identificate nella precedente tabella di funzione.

Per quanto riguarda la tabella di funzione, i simboli sono usati come segue:

X rappresenta un "non importa"

↓ rappresenta una transizione logica da 1 a 0

↑ rappresenta una transizione logica da 0 ad 1

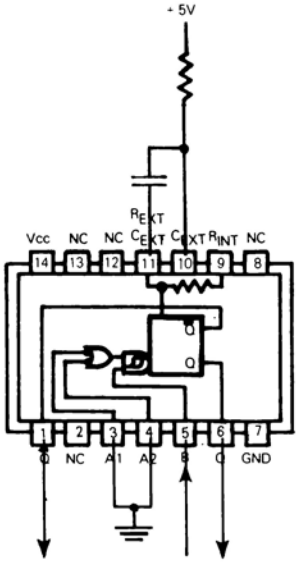
⎍ rappresenta un one-shot con un livello logico monostabile ed un livello d'impulso uno

⎎ è il NOT di ⎍

La durata dell'uscita one-shot è determinata da una rete resistenza-capacità proprio come descritta per il multivibratore monostabile Signetics 555; ma ci sono alcune differenze. Il 74121 fornisce un resistore interno al quale si può accedere connettendo R_{INT} (pin 9) a V_{CC} (pin 14). Un resistore esterno variabile può essere connesso tra R_{INT} (pin 9) oppure R_{EXT} (pin 11) e V_{CC} (pin 14).

Un condensatore di timing esterno, se presente, sarà connesso tra C_{EXT} (pin 10) ed R_{EXT} (pin 11).

Questo è un modo in cui il multivibratore monostabile 74121 può essere connesso:



L'impiego del multivibratore monostabile 74121 corrisponde alle ultime due righe della tabella di funzione.
Una rete esterna resistore/condensatore controlla la durata dell'impulso one-shot. Ogni impulso one-shot sarà comandato da una transizione da basso ad alto al pin 5 (B).

Dal punto di vista della programmazione ci sono solo due caratteristiche significative del multivibratore monostabile 74121:

1) **Le uscite monostabili sono equivalenti ai digit binari di valore fissato.** Qualsiasi istruzione immediata che carica uno zero od un uno in qualsiasi bit del registro simula l'uscita monostabile. Per esempio:

```
MVI     B,4     ; Pone il bit 3 del Registro B ad 1, ripristina tutti gli
                ; altri bit
```

   Il bit 3 del Registro B è equivalente ad un flip-flop; così ogni altro bit del Registro B ed ogni altro registro.

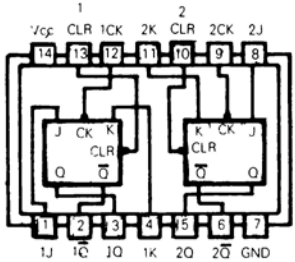
2) **Una uscita one-shot assume un ritardo di tempo di valore fissato.** Si mostrerà come questo ritardo di tempo può essere calcolato all'interno del sistema microcalcolatore ma prima si esamina il flip-flop master-slave 74107.

IL FLIP-FLOP 74107 DUALE J-K MASTER-SLAVE CON CLEAR

Si consideri il flip-flop master-slave 74107. Questo flip-flop è illustrato come segue:

TABELLA DI FUNZIONE

INGRESSI				USCITE	
1CLR o 2CLR	1CK o 2CK	1J o 2J	1K o 2K	1Q o 2Q	1$\overline{Q}$ o 2$\overline{Q}$
L	X	X	X	L	H
H	⎍	L	L	Permane com'era	
H	⎍	H	L	"	L
H	⎍	L	H	L	H
H	⎍	H	H	Cambia stato indipendentemente da quello precedente	



⎍ identifica un impulso di clock; il modo in cui esso è impiegato è descritto di seguito.

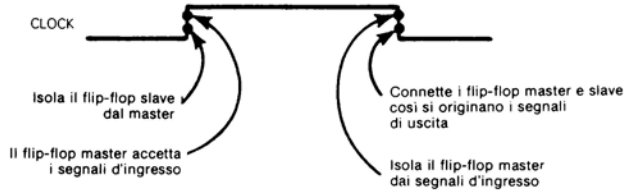
X significa "non importa"

Si esamini la tabella di funzione sopra illustrata. Se non si è familiari con questo tipo di dispositivo logico le sue caratteristiche non sono del tutto evidenti.

FLIP-FLOP MASTER-SLAVE

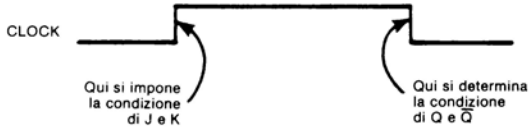
La notazione "master-slave" identifica un circuito che ha, infatti, due flip-flop. Perciò esistono quattro flip-flop nel dispositivo 74107 sopra illustrato.

I flip-flop in ogni coppia master-slave rispondono in questo modo ad un segnale di clock:

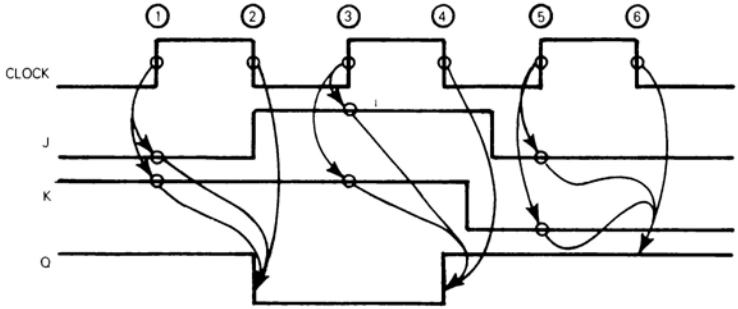


Il significato di questa risposta al segnale di clock è che gli ingressi del flip-flop devono essere presenti al gradino positivo del segnale di clock; questi ingressi devono rimanere stazionari mentre il segnale di clock è alto. Le uscite del flip-flop, comunque, non cambiano stato finchè non si presenta il gradino negativo del segnale di clock.

Il segnale di clock può essere impiegato per originare i ritardi di tempo. L'uscita del flip-flop 74107 è determinata dai livelli del segnale d'ingresso esistenti qualche periodo di tempo prima. Questo può eseere illustrato come segue:



Ecco un esempio specifico



La descrizione che segue di questo diagramma di timing fa riferimento ai numeri indicati sul segnale di clock.

All'istante ② l'uscita Q diventa bassa poichè ad ① J era basso e K alto.

All'istante ④ Q cambia stato perchè a ③ J e K erano entrambi bassi.

All'istante ⑥ Q resta inalterata poichè a ⑤ J e K erano entrambi bassi.

SIMULAZIONE AL MICROCALCOLATORE IN TEMPO REALE

Qual'è il significato del multivibratore monostabile 555 e dei flip-flop master-slave? Quando si esegue la simulazione al microcalcolatore di questi dispositivi c'è solo una caratteristica importante da discutere — è il concetto di tempo reale.

Il multivibratore monostabile 555 crea impulsi di livello logico alto alla sua uscita dove la durata del livello logico alto è una funzione controllabile in tempo reale.

Il flip-flop master-slave 74107 permette la generazione di un segnale d'uscita basato sulle condizioni dell'ingresso di qualche tempo prima.

CICLI DI ISTRUZIONI DI TIMING DEL MICROCALCOLATORE

BREVI INTERVALLI DI TEMPO DI TIMING

E' abbastanza semplice creare un ritardo di tempo impiegando un sistema a microcalcolatore — provvedendo che il sistema a microcalcolatore non sia impiegato per eseguire nessuna altra operazione simultanea. Si consideri la seguente sequenza di istruzione:

```
Cicli
                MVI     A, TIME   ; Carica la costante di tempo nell'Accumulatore
5       LOOP    DCR     A         ; Decrementa l'Accumulatore
10              JNZ     LOOP      ; Decrementa ulteriormente se non zero
```

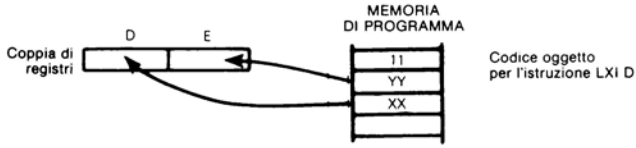
La precedente sequenza d'istruzione carica un valore dati, rappresentato dalla label TIME, nell'Accumulatore. L'Accumulatore è decrementato fino a zero ed in quello istante continua l'esecuzione del programma. Si assuma che sia impiegato un clock di 500 nanosecondi dal sistema microcalcolatore. Le istruzioni DCR e JNZ, considerate insieme, si eseguono in 15 cicli-equivalenti a 7,5 microsecondi. Questo significa che la sequenza del programma precedentemente illustrato può originare un ritardo da un valore minimo di 7,5 microsecondi (quando TIME è uguale a 1) ed, aumentando di 7,5 microsecondi alla volta, fino ad un massimo ritardo di 1920 microsecondi, che sono equivalenti a 7,5 x 256. Il massimo ritardo di tempo risulterà quando TIME ha un valore iniziale zero poichè TIME è decrementato PRIMA del controllo per vedere se è zero; perciò questo tempo è determinato decrementando 1 a 0 e non 0 ad FF_{16}.

LUNGHI INTERVALLI DI TEMPO DI TIMING

Ritardi di tempo lunghi possono essere generati mediante un contatore a 16-bit. Ecco un'opportuna sequenza di istruzioni:

```
Cicli
                LXI     D,T16     ; Carica la costante di tempo in D ed E
5       LOOP    DCX     D         ; Decrementa D E
5               MOV     A,D       ; Prova per zero tramite OR
4               OR                ; Contenuti di D ed E attraverso l'Accumulatore
10              JNZ     LOOP
```

L'istruzione LXI carica un valore a 16-bit, rappresentato dalla label T16, nella coppia di registri DE. L'istruzione LXI, essendo un'istruzione immediata, origina tre bytes di codice oggetto. Quando viene eseguita l'istruzione LXI succede questo:



CONVALIDA DI STATO IMPIEGANDO L'ISTRUZIONE DCX

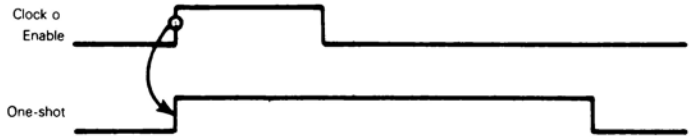
L'istruzione DCX decrementa il valore a 16-bit dei registri DE considerati una singola entità dati. Comunque una stranezza del set di istruzione 8080 trascura di porre i bit di stato basati sul risultato del decremento a 16 bit. Questo significa che non esiste un modo immediato di conoscenza se i registri DE in un certo istante contengono zero o no. Per fare questa prova si caricano i contenuti del registro D nell'Accumulatore, quindi si opera l'OR con i contenuti del registro E. Se il risultato dell'Accumulatore è zero allora entrambi i registri D ed E devono contenere 0. Se il risultato non è zero si ritorna a decrementare il valore a 16-bit.

Si osservi che sono necessari 24 cicli per percorrere una volta un ciclo di istruzione a lungo intervallo di tempo. Assumendo anche qui che il microcalcolatore sia pilotato da un clock di 500 nanosecondi, esso impiegherà 12 microsecondi per eseguire una volta il ciclo di istruzioni. Il valore minimo che T16 può avere è 1. Il valore massimo è ancora 0 perchè il decremento è fatto prima del controllo per zero; quindi inizialmente sarebbe caricato 0 nei registri D ed E e sarebbe decrementato ad $FFFF_{16}$ prima di eseguire il primo test per zero. Così il ciclo d'istruzione per lunghi intervalli di tempo genererà ritardi che variano con incrementi di 12 microsecondi da un minimo di 12 microsecondi ad un massimo di 0,786432 secondi.

$FFFF_{16} = 65536_{10}$
12 x 65536 = 786432 microsecondi

INIZIALIZZAZIONE DEL RITARDO DI TEMPO

Ora la simulazione effettiva di uno one-shot è complicata dal fatto che si possono calcolare i ritardi di tempo ma quando questi iniziano? Per i dispositivi a logica digitale la risposta è semplice: il ritardo di tempo inizia quando cambia stato il segnale d'ingresso:

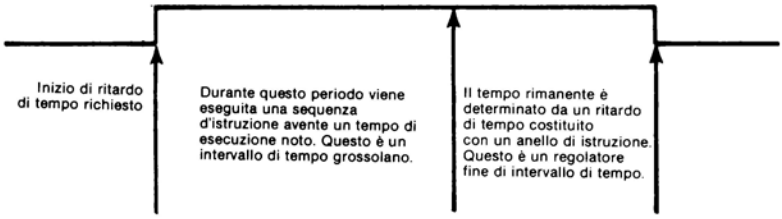


Per fare il parallelo di questo concetto all'interno di un programma al microcalcolatore si deve iniziare un ritardo di tempo dal completamento di qualche altra esecuzione di sequenza di programma. Questo concetto può essere illustrato come segue:

```
        -
        -
        -
        JMP     DELY      ; Ultima istruzione di qualche sequenza precedente
        -
        -
        -
DELY    MVI     A,TIME    ; Istruzione di breve intervallo di tempo
LOOP    DCR     A         ; Sequenza
        JNZ     LOOP
```

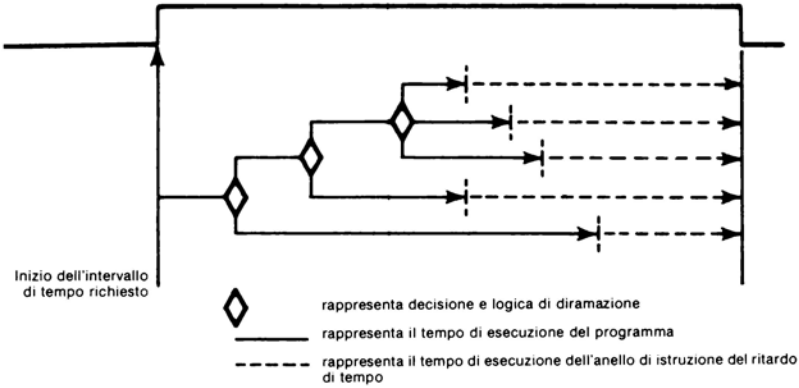
ESECUZIONE DI PROGRAMMI ALL'INTERNO DI RITARDI DI TEMPO

Esiste un altro problema associato alla generazione di ritardi di tempo all'interno di un sistema a microcalcolatore impiegando i cicli di esecuzione di istruzione come descritti: il microcalcolatore, in sostanza, non esegue un lavoro costruttivo durante questo ritardo. Può esistere un semplice rimedio a questo problema provvedendo a definire un programma per il microcalcolatore da eseguire durante il periodo del ritardo di tempo. Questo può essere così illustrato:



Si deve assumere di poter calcolare il tempo esatto che impiegherà il programma per eseguire il ritardo di tempo all'interno dell'one-shot; inoltre il tempo calcolato deve essere minore od uguale al ritardo di tempo.
Non molti programmi approssimeranno questa descrizione. Se, per esempio, la maggior parte della sequenza d'istruzione può essere eseguita, in dipendenza delle condizioni attuali, allora per l'esecuzione di un programma può essere richiesto un numero di volte molto diverso. Inoltre come esiste un numero fisso di diramazioni identifica-

bili, il problema è trattabile e può essere illustrato come segue:



Ora ogni ramo delle diramazioni del programma terminerà come segue:

```
        MVI     A,DLY 1   ; Carica il primo ritardo di tempo
        JMP     LOOP      ; Inizia il ciclo del ritardo di tempo
        -
        -
        -
        MVI     A,DLY 2   ; Carica il secondo ritardo di tempo
        JMP     LOOP      ; Inizia il ciclo del ritardo di tempo
        -
        -
        -
        MVI     A,DLY 3   ; Carica il terzo ritardo di tempo
        JMP     LOOP      ; Inizia il ciclo del ritardo di tempo
        -
        -
        -
        MVI     A,DLY 4   ; Carica il quarto ritardo di tempo
        JMP     LOOP      ; Inizia il ciclo di ritardo di tempo
        -
        -
        -
        MVI     A,DLY 5   ; Carica il quinto ritardo di tempo
        JMP     LOOP      ; Inizia il ciclo del ritardo di tempo
        -
        -
        -
LOOP    DCR     A         ; Istruzione dell'intervallo di tempo breve
        JNZ     LOOP      ; Sequenza
```

E' molto comune per un programma di microcalcolatore il contenere numerose diramazioni condizionali; possono esistere quindi centinaia di possibili tempi di esecuzione dipendentemente dalle varie combinazioni delle condizioni attuali. L'esecuzione di un programma all'interno dell'intervallo di tempo del ritardo richiesto ora diventa impraticabile a causa della logica richiesta per calcolare il tempo rimanente per le innumerevoli diramazioni del programma che è proprio troppo complicata.

I LIMITI DELLA SIMULAZIONE DELLA LOGICA DIGITALE

Un microcalcolatore del tipo 8080 può calcolare i ritardi di tempo mentre nessun altro programma richiede di essere eseguito durante il ritardo di tempo oppure fornendo una sequenza d'istruzione molto semplice con diramazioni molto limitate che viene eseguita durante il ritardo di tempo.

RITARDI DI TEMPO SIMULTANEI

Non si possono simulare ritardi di tempo simultanei oppure un ritardo di tempo che deve ricorrere in parallelo all'esecuzione di un programma non ben definito. La logica esterna deve manipolare tutti questi ritardi di tempo.

REALIZZAZIONE DELL'INTERFACCIA CON ONE-SHOTS ESTERNI

Si noti che anche se la logica esterna può dover realizzare dei ritardi di tempo, è molto facile per il sistema microcalcolatore comandare l'inizio del ritardo di tempo e per la logica esterna riportare il completamento del ritardo di tempo.

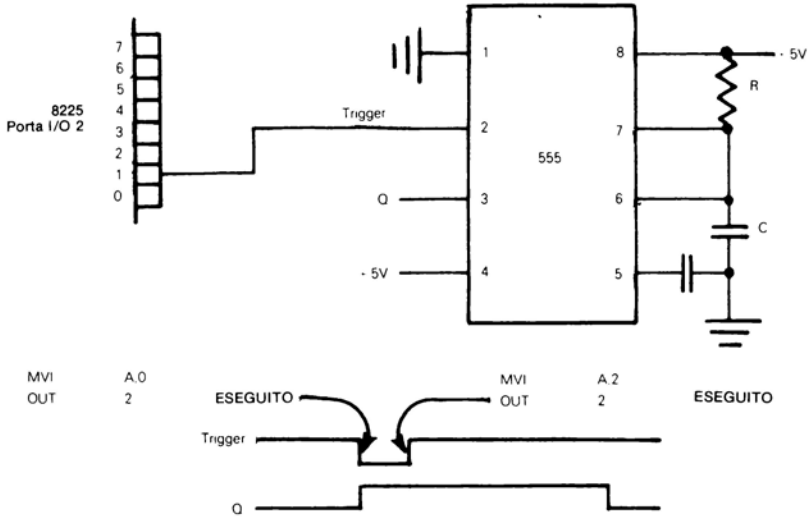
INIZIALIZZAZIONE ONE-SHOT

Si può identificare l'inizio del ritardo di tempo semplicemente emettendo un appropriato digit binario. Si osservi ancora che la via "Uscita del Segnale" era l'uscita alla logica esterna mediante la simulazione dell'invertitore di segnale. L'uscita di un segnale alla logica esterna e davvero molto semplice. Si considerino le seguenti quattro istruzioni:

```
MVI   A,0    ; Carica uno 0 nell'Accumulatore
OUT   2      ; Uscita attraverso la Porta I/O 2
MVI   A,2    ; Carica un 1 nel bit 1 dell'Accumulatore
OUT   2      ; Uscita attraverso la Porta I/O 2
```

Un 1 è uscito al pin 1 della porta I/O 2. Assumendo che il pin associato con questa porta I/O sia connesso al trigger di un multivibratore e che questa connessione sia precedentemente al livello alto, allora la semplice esecuzione delle precedenti istruzioni farà scattare uno one-shot.

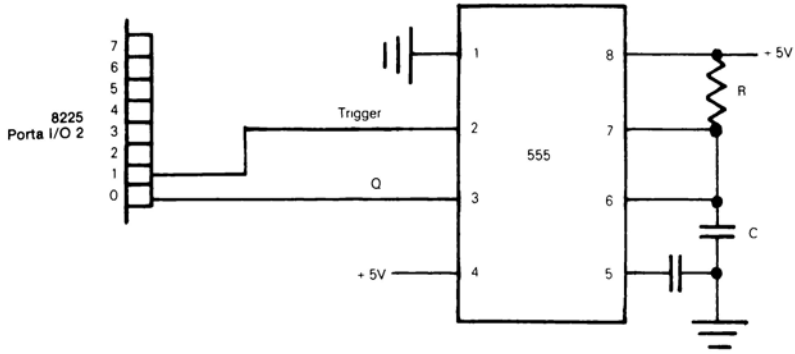
Questo può essere illustrato come segue:



E' altrettanto facile per la logica esterna segnalare la fine del ritardo di tempo.

ONE-SHOT TIME OUT IMPIEGANDO LO STATO

Se si sta trattando con la logica "maggiore od uguale a" tutto quanto necessario per un'uscita one-shot deve essere connesso ad un altro pin di una porta I/O del microcalcolatore:



I segnali che arrivano ai pin delle porte I/O passano attraverso buffers. Il programma da eseguire mediante il microcalcolatore può, in ogni istante, far entrare i contenuti della porta I/O e convalidare la condizione del bit 0 che è stato cablato con l'uscita Q. Quando si trova che questo bit è uguale a 0, la logica del programma del microcalcolatore deduce che l'intervallo di tempo è stato superato.

La sequenza di istruzione seguente convaliderà la porta I/O, ripristinerà l'ingresso one-shot ed azzererà lo stato "intervallo di tempo completo" corrispondente al pin 0 della Porta I/O 2:

```
        IN      2       ; Ingresso dei contenuti della Porta I/O 2 all'Accu-
                        ; mulatore
        ANI     1       ; Maschera tutti i bit tranne il bit 0
        JNZ     NEXT    ; Continua se il bit è 1
; Il programma TIME OUT comincia qui
        —
        —
        —
NEXT                    ; Il programma NOT TIME OUT inizia qui
```

L'istruzione IN muove i contenuti attuali della Porta I/O 2 all'Accumulatore.

La seguente istruzione ANI pone a 0 tutti i bit dell'Accumulatore eccetto il bit corrispondente alla Porta I/O 2, pin 0:

```
7 6 5 4 3 2 1 0 ← N. Bit
X X X X X X X Y   Contenuti accumulatore
0 0 0 0 0 0 0 1   Esadecimale 01
-----------------
0 0 0 0 0 0 0 Y   Risultato dell'AND
```

Se l'ingresso del digit binario dal pin 0 della Porta I/O 2 è 1 allora l'uscita Q è ancora alta. L'istruzione JNZ NEXT continua semplicemente l'esecuzione del programma.

Se il bit 0 della Porta I/O 2 è 0 allora il ritardo di tempo è finito; si dirotta su una sequenza del programma che accetta solo esecuzioni immediatamente seguenti un time out.

TIME OUT ED INTERRUPTS

L'esatto termine di un time out può essere segnalato al microcalcolatore impiegando un interrupt.

Ora non appena si verifica un one-shot time out, esso forza il sistema microcalcolatore a cessare l'esecuzione di qualsiasi programma in corso di esecuzione. Sarà forzato un dirottamento a qualche altro programma che è stato specificatamente assegnato per rispondere al time out.

Le considerazioni di programmazione associate con gli interrupts sono molto più complicate rispetto al livello ottenuto con il Capitolo 2. Si differirà perciò una dettagliata descrizione dell'elaborazione di interrupt nel corso del libro. Per il momento è sufficiente capire che l'istante esatto di un time out può essere segnalato al microcalcolatore impiegando la logica dell'interrupt.

Capitolo 3

UNA SIMULAZIONE DIRETTA DELLA LOGICA DIGITALE

I dispositivi logici discreti simulati al Capitolo 2 non sono stati selezionati a caso; posizionati correttamente in sequenza essi simuleranno la logica illustrata in Figura 3-1. Questa logica è una parte dell'interfaccia di stampante per le stampanti Qume delle serie Q e Sprint. La Figura 3-2 è un diagramma di timing con riferimento alla Figura 3-1. Si descriveranno entrambe le figure ad un livello molto elementare.

Ora lo scopo di questo capitolo è di fornire una correlazione uno ad uno tra la programmazione in linguaggio assembly del microcalcolatore e del progetto logico digitale. Quello che si deve capire è che, mentre tale correlazione uno ad uno può essere forzata, essa non è naturale; e questo è dove il problema consente la comprensione. I programmi del microcalcolatore dovrebbero essere scritti per sottolineare la natura dei microcalcolatori e non le caratteristiche della logica digitale.

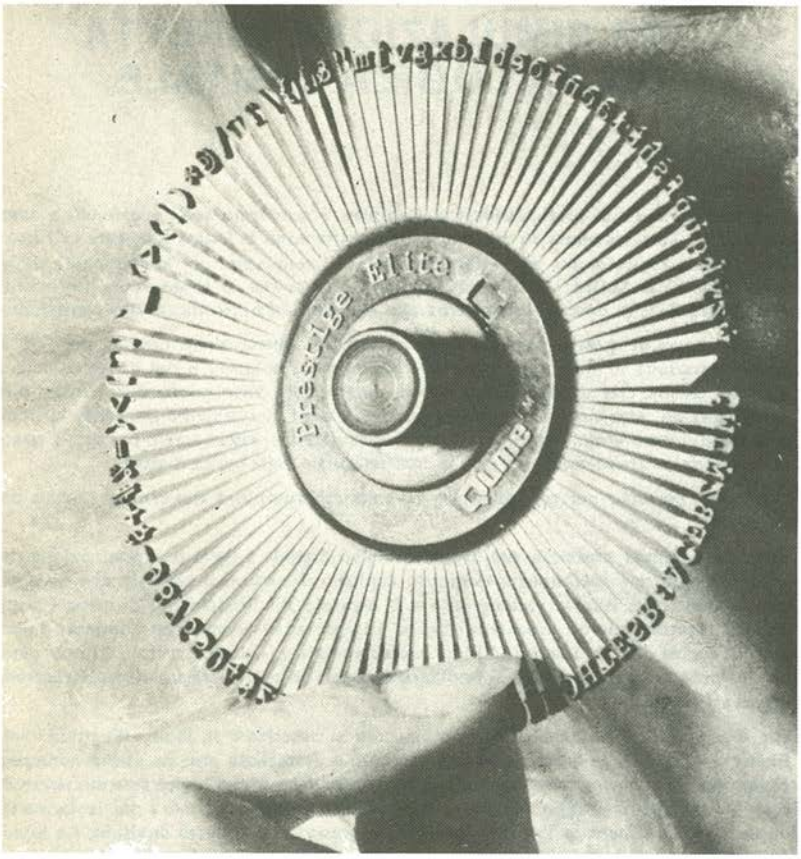
Un modo corretto per programmare un microcalcolatore è descritto a partire dal Capitolo 4.

Tuttavia la giusta posizione del progetto logico digitale e della programmazione del microcalcolatore è sottolineata in questo capitolo. Questo è il capitolo che collega i due concetti; per questa ragione esso è il capitolo più importante di questo libro. Questo capitolo per un progettista logico è importante perchè esso eliminerà i concetti di logica digitale inapplicabili ai microcalcolatori. Questo capitolo è importante per un programmatore perchè si familiarizzerà con un nuovo traguando-realizzazione logica efficiente.

Per ottenere il traguando di questo capitolo si descriverà la logica illustrata nelle Figure 3-1 e 3-2; la descrizione sarà accurata e dettagliata così da essere comprensibile non solo ai progettisti logici. Nel caso della descrizione logica saranno riportati casi semplici di linguaggio assembly. Se si conosce la logica digitale è particolarmente importante confinare la lettura alle parti in grassetto di questo capitolo. La logica della Figura 3-1 è descritta in dettaglio sufficiente per soddisfare le richieste di un programmatore, un lettore senza basi di logica.

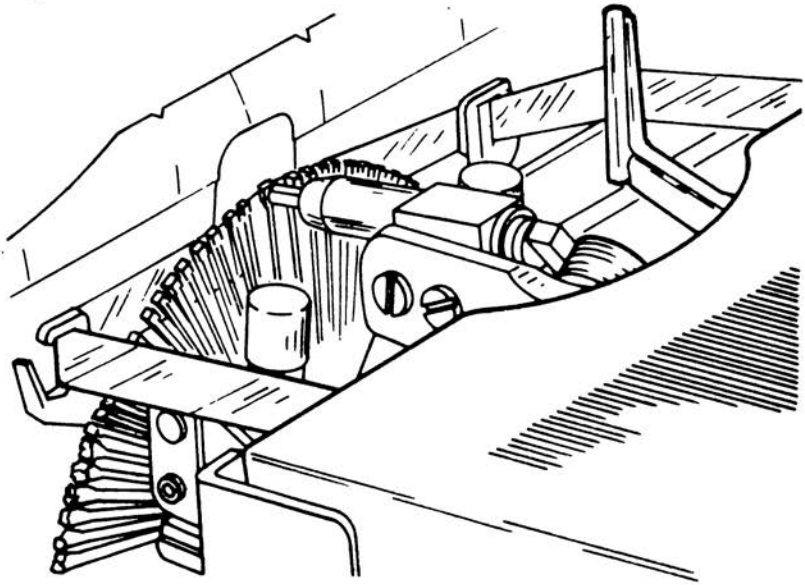
COME FUNZIONA LA STAMPANTE QUME

L'elemento attivo della stampante Qume è una ruota di stampa a 96-petali con un carattere ogni petalo.

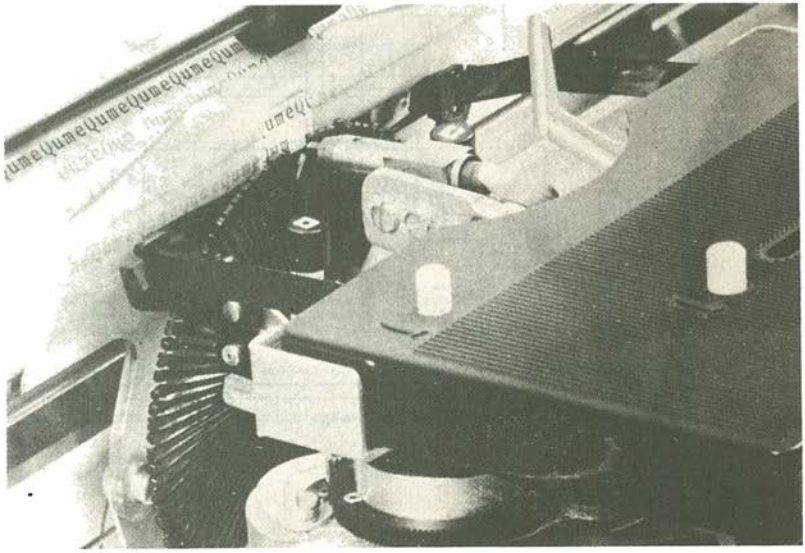


Per gentile concessione della QUME CORPORATION

Un carattere viene stampato muovendo la ruota di stampa finchè il petalo appropriato è di fronte al martelletto di stampa comandato da un solenoide. Il martelletto di stampa viene azionato; esso batte contro il petalo della ruota di stampa che contrassegna la carta:



Ogni volta che un carattere non è in fase di stampa, la ruota di stampa è posizionata con un petalo corto verticale cosicchè il carattere appena stampato è visibile:



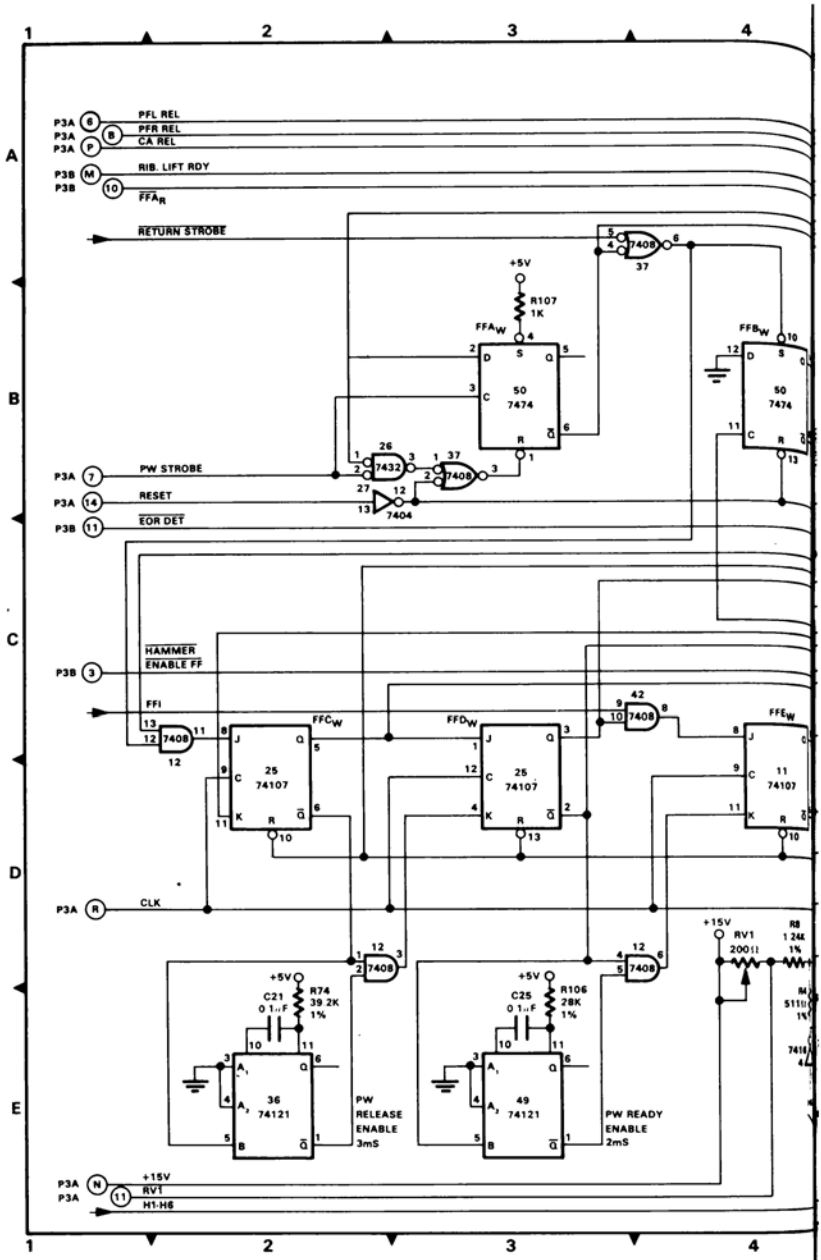
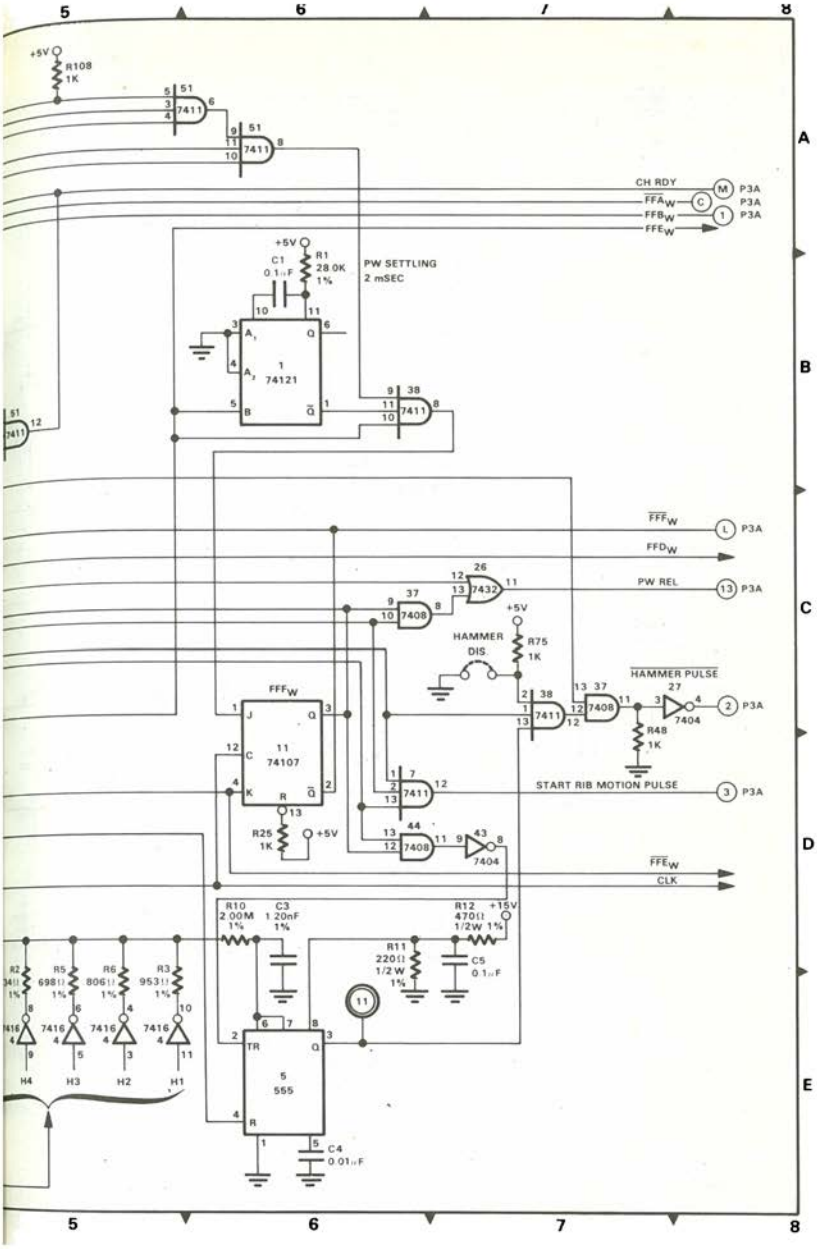


Figura 3-1. Logica di controllo della ruota di stampa.


5
6
7
8
A
B
C
D
E
+5V
R108
1K
51
7411
CH RDY
FFA_W
FFB_W
FFE_W
M
C
1
P3A
+5V
C1
0.1μF
R1
28.0K
1%
PW SETTLING
2 mSEC
1
74121
38
7411
FFF_W
FFD_W
L
PW REL
13
26
7432
37
7408
+5V
HAMMER
DIS.
R75
1K
HAMMER PULSE
27
7404
2
R48
1K
FFF_W
11
74107
START RIB MOTION PULSE
3
R25
1K
+5V
44
7408
43
7404
FFE_W
CLK
R10
2.00M
1%
C3
1.20nF
1%
R12
470Ω
1/2W 1%
+15V
R11
220Ω
1/2W
1%
C5
0.1μF
11
R2
R5
698Ω
1%
R6
806Ω
1%
R3
953Ω
1%
7416
H4
H3
H2
H1
5
555
C4
0.01μF

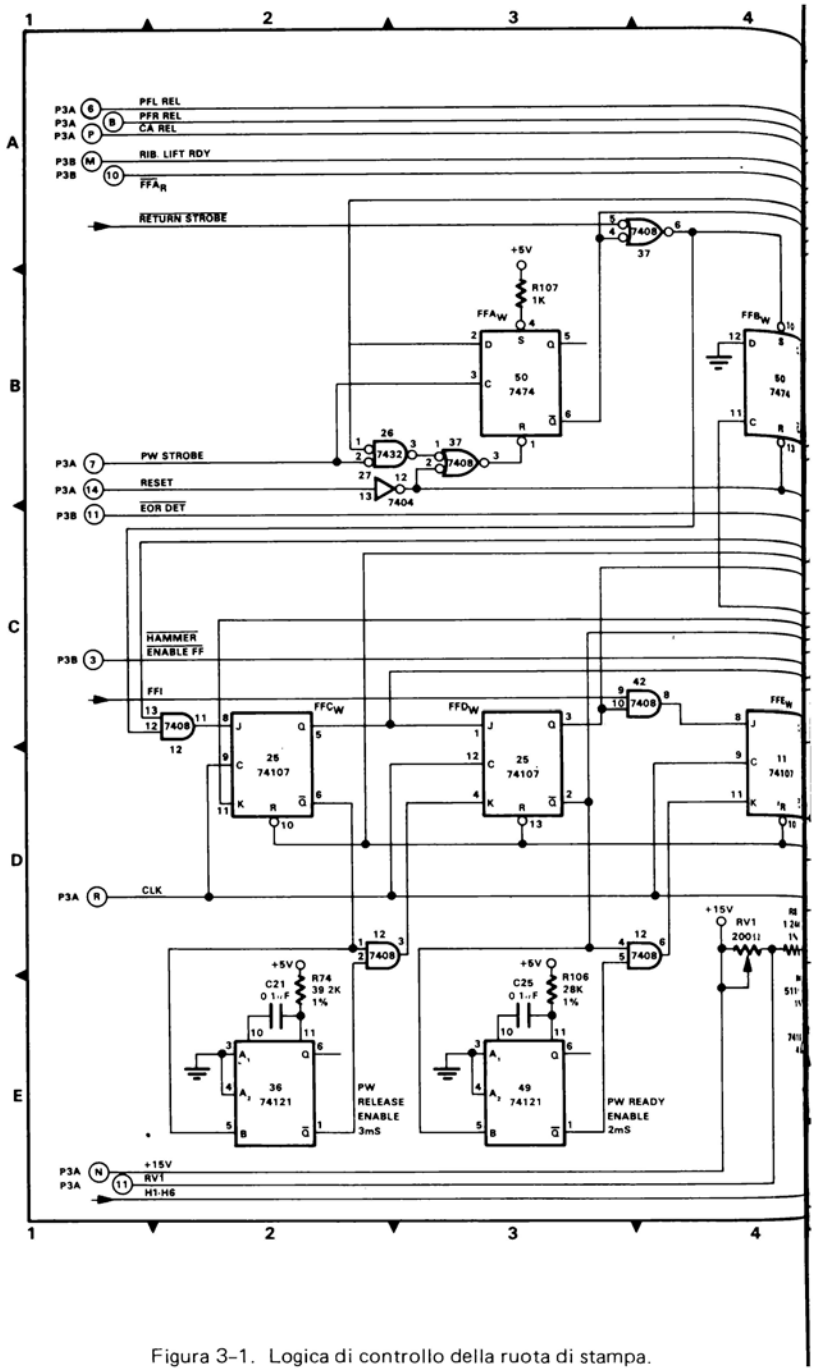
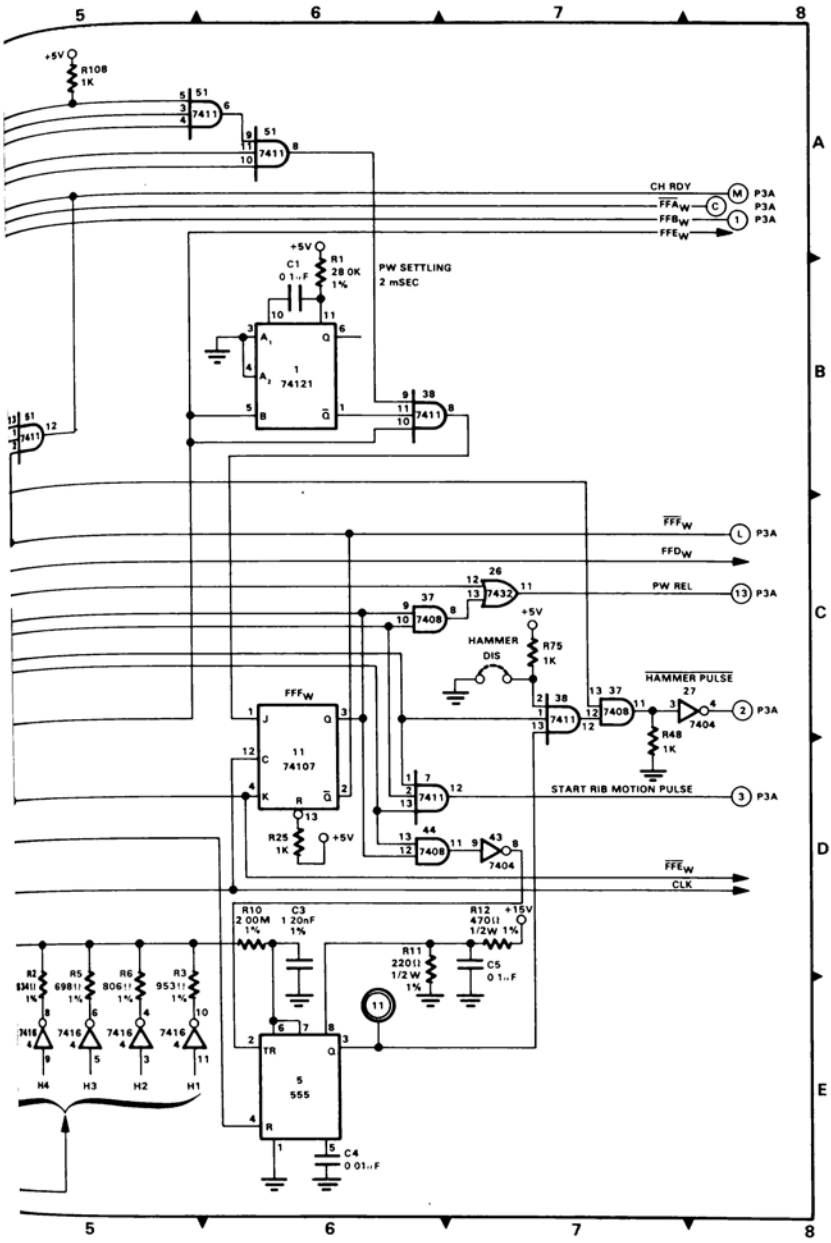


Figura 3-1. Logica di controllo della ruota di stampa.


5
6
7
8
A
B
C
D
E
+5V
R108
1K
7411
51
CH RDY
FFA_W
FFB_W
FFE_W
P3A
PW SETTLING
2 mSEC
R1
28 0K
1%
C1
0 1µF
74121
38
FFF_W
FFD_W
PW REL
7432
26
7408
37
HAMMER
DIS
R75
1K
HAMMER PULSE
7404
27
R48
1K
74107
11
R25
1K
START RIB MOTION PULSE
44
43
CLK
R10
2 00M
1%
C3
1 20nF
1%
R12
470Ω
1/2W 1%
+15V
R11
220Ω
1/2W
1%
C5
0 1µF
R2
R5
R6
R3
7416
H4
H3
H2
H1
555
C4
0 01µF

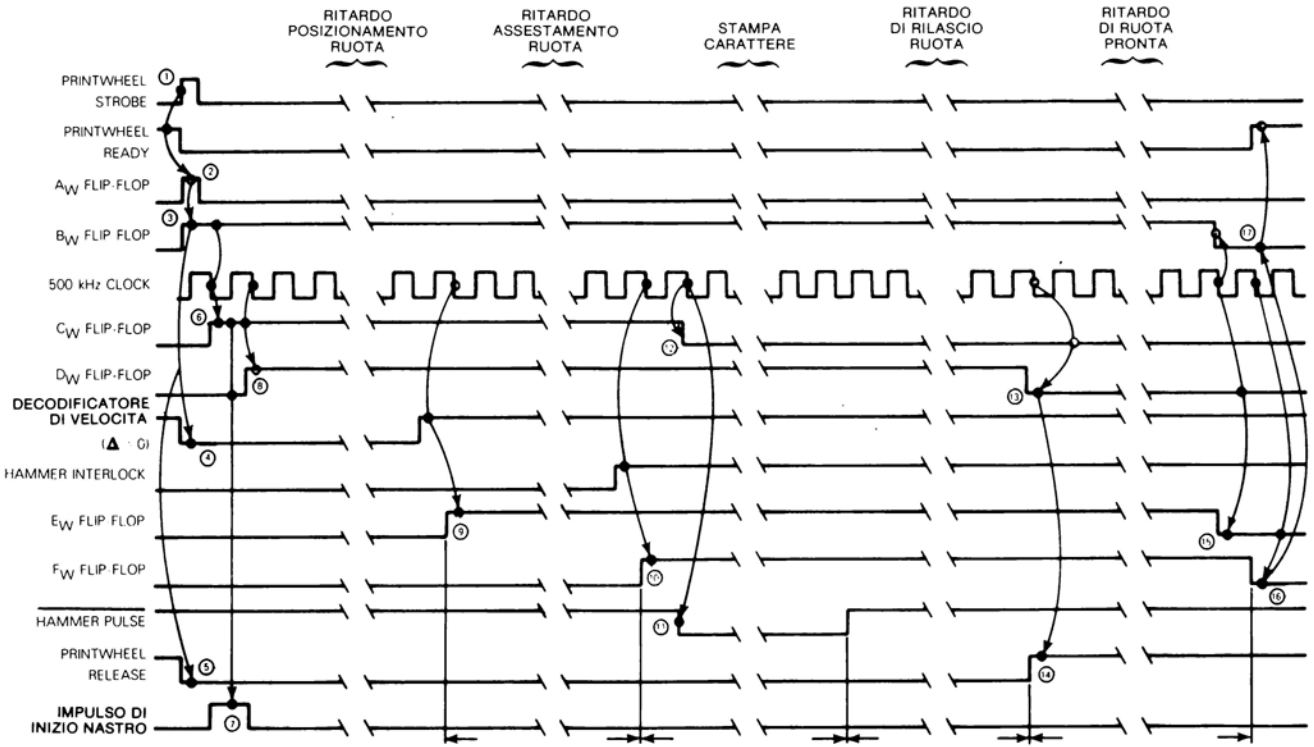


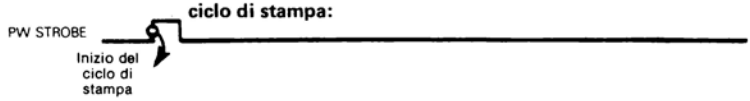
Figura 3–2. Diagramma del timing della logica di controllo della ruota di stampa

Come parte del ciclo di stampa devono essere mossi il nastro stampante ed il carrello della carta.

Ogni carattere è stampato secondo una definita sequenza di eventi, l'insieme dei quali è definito "ciclo di stampa". La logica illustrata in Figura 3-1 controlla il carattere del ciclo di stampa. **Questi sono gli eventi che devono verificarsi in un ciclo di stampa:**

PW STROBE

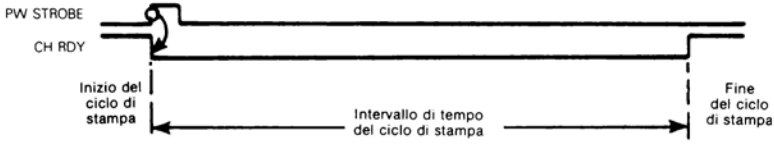
1) Primo, il ciclo di stampa deve essere inizializzato. Un **segnale (PW STROBE) è pulsato al livello alto per inizializzare il ciclo di stampa:**



RUOTA DI STAMPA PRONTA

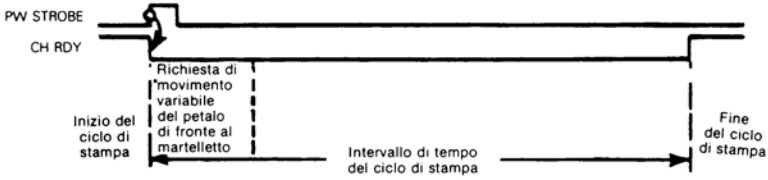
CH RDY

2) Il ciclo di stampa impiegherà un intervallo di tempo fissato. Ovviamente durante questo periodo di tempo un altro ciclo di stampa non deve essere inizializzato. **Perciò la logica esterna,** responsabile della generazione del segnale PW STROBE **deve fornire un segnale di identificazione della durata del ciclo di stampa. Questo segnale è RUOTA DI STAMPA PRONTA, chiamato anche CH RDY:**



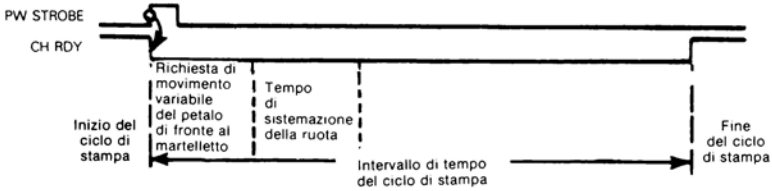
La sequenza degli eventi che effettivamente genera la stampa di un carattere può ora procedere con la sicurezza che la logica esterna non tenterà di iniziare la stampa di un nuovo carattere prima che sia stato completato l'attuale ciclo di stampa.

3) **La ruota di stampa è mossa dalla sua posizione di visibilità finchè il petalo del carattere appropriato è di fronte al martelletto di stampa:**



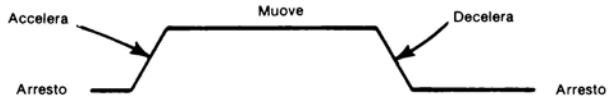
Un ritardo di tempo variabile è richiesto dalla logica di posizionamento della ruota di stampa. Ovviamente esso sarà maggiore per posizionare un petalo che è lontano dalla posizione di visibilità di quello per posizionare un petalo adiacente.

4) Prima che venga azionato il martelletto, **la ruota deve avere il tempo per posizionarsi.** Un ritardo di tempo fisso di 2 millisecondi è sufficiente.

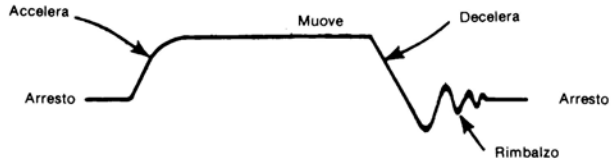


RITARDI DI POSIZIONAMENTO

I ritardi di tempo di posizionamento sono un aspetto molto importante della logica di supporto di qualunque tipo di movimento meccanico. E' facile tracciare una linea netta che mostri la velocità di movimento come segue:



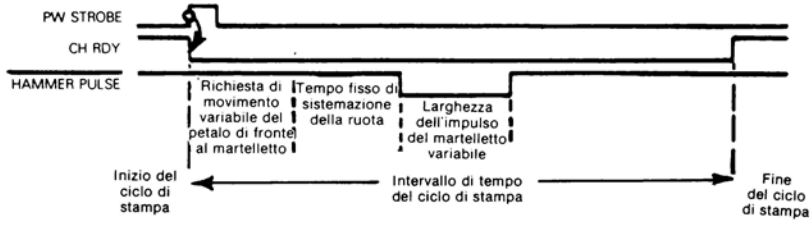
Ma in realtà il movimento che si verifica è il seguente:



Il transitorio che segue la decelerazione deve essere coperto da un ritardo di tempo di posizionamento.

Verrà stampato un carattere macchiato se la ruota di stampa sta ancora vibrando quando il martelletto spinge il petalo contro la carta.

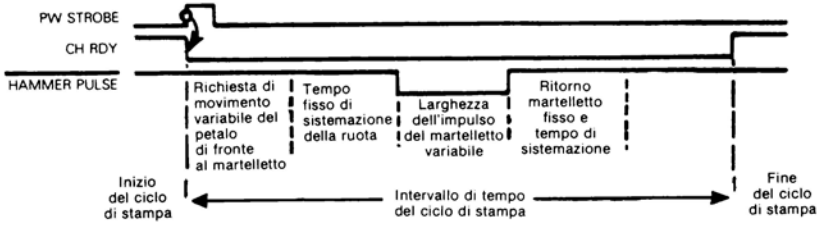
5) Alla fine del tempo di ritardo per il posizionamento della ruota di stampa **il martelletto può essere azionato.** Questo viene eseguito inviando un impulso al solenoide. **Vengono fornite sei intensità di impulsi di comando del martelletto** poichè alcuni caratteri hanno un'area diversa da altri. Per premere un'area relativamente grande come quella di un "W" con la stessa intensità con cui si preme un piccolo carattere come "." impiegando la stessa forza si avrebbero disuniformità nella densità del testo stampato. La durata dell'impulso che alimenta il solenoide è controllata dal ritardo di tempo successivo.



Il tratto sopra $\overline{\text{HAMMER PULSE}}$ identifica che il segnale è attivo quando è al livello basso.

6) Al termine dell'impulso di ritardo di tempo, **il martelletto ha colpito un petalo e lo ha**

forzato sulla carta. Un ritardo di 3 millisecondi viene generato per questo scopo:



POSIZIONE DI VISIBILITA' DELLA RUOTA DI STAMPA

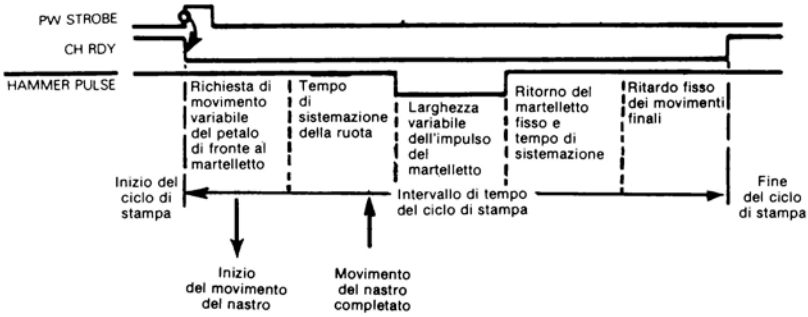
7) **Ora la punta di stampa può essere mossa alla sua posizione di visibilità** ed il carrello della carta può essere avanzato alla posizione del carattere successivo. La "posizione di visibilità" della ruota di stampa normalmente è una posizione non attiva; in questa posizione un petalo corto è di fronte al martelletto di stampa cosicchè l'ultimo carattere stampato è visibile sopra il petalo corto, per questo si chiama "posizione di visibilità". Se non si dà tempo al martelletto per riposizionarsi prima del movimento della ruota alla sua posizione di visibilità, si può rompere un petalo della ruota poichè il martelletto picchierà alla sommità del petalo sporgente. Inoltre la carta può essere macchiata dall'urto del petalo impiegato. Poichè il martelletto impiega un certo tempo per ritrarsi completamente, non sorgerà nessuno di questi problemi.

Un ritardo di tempo finale di 2 millisecondi consente il riposizionamento della ruota e del carrello della carta:

IMPULSO D'INIZIO DEL NASTRO
FF A

8) E per quanto riguarda la logica del nastro? **Allo scopo di ottenere una nitida impressione della carta, un tratto di nastro fresco deve essere presente tra il petalo del carattere e la carta.** Brevemente, dopo l'inizio del ciclo di stampa, perciò, un segnale (IMPULSO DI INIZIO DEL MOVIMENTO DEL NASTRO) viene inviato alla logica esterna che effettivamente controlla il movimento del nastro. Questa logica esterna (non compresa nella Figura 3–1) riporta un segnale di movimento del nastro eseguito ($\overline{FFA}$) poichè non si può consentire l'azionamento del martelletto se il nastro è ancora in movimento.

Così **il nastro viene avanzato mentre la ruota viene posizionata inizialmente:**



In conclusione, un ciclo di stampa consiste di 5 ritardi di tempo; ogni ritardo di tempo fa partire una complicata attività logica seguita da un periodo di movimento meccanico.

SEGNALI D'INGRESSO ED USCITA

A questo stadio si ha una conoscenza generale delle funzioni controllate dalla logica della Figura 3-1; **la fase successiva è di osservare più da vicino i segnali d'ingresso e d'uscita.**

Allo scopo di conoscere cosa fare e quando farlo si deve fare affidamento interamente sui segnali d'ingresso. Analogamente i segnali d'uscita rappresentano il solo modo in cui si può trasmettere il controllo alla logica esterna.

Ci si propone a questo punto un traguardo limitato; la comprensione di quale funzione esegue ogni segnale d'ingresso ed uscita e come — fisicamente — si possono manipolare i segnali. Si discuterà prima quest'ultimo problema.

DISPOSITIVI INGRESSO/USCITA

INTERFACCIA PERIFERICA PROGRAMMABILE

Ci sono due tipi di dispositivi impiegati per trasmettere i segnali ed i dati tra un sistema a microcalcolatore 8080 e la logica esterna.

Primo, ci sono dispositivi d'interfaccia periferica programmabile come la famiglia di dispositivi 8255 prodotti da diverse industrie oppure il TMS 5501 prodotto dalla Texas Instruments.

BUFFER DI TIPO LATCH

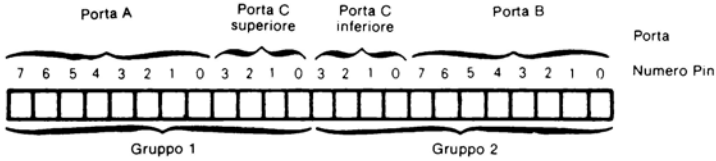
Quindi ci sono i buffers di tipo latch come l'8212.

Si impiegherà l'8255 Interfaccia Periferica Programmabile ed un 8212 buffer di tipo latch.

Poichè questi dispositivi sono stati descritti in "An Introduction To Microcomputers" si assumeranno note le loro caratteristiche ed organizzazione generali. Se queste non sono note si consulti An Introduction To Microcomputers - Volume II - Some Real Products prima di continuare la seguente discussione.

L'INTERFACCIA PERIFERICA PROGRAMMABILE 8255

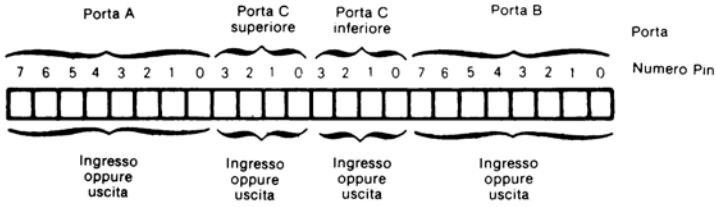
L'Interfaccia Periferica Programmabile 8255 fornisce 24 pins I/O che possono essere raggruppati in porte come segue:



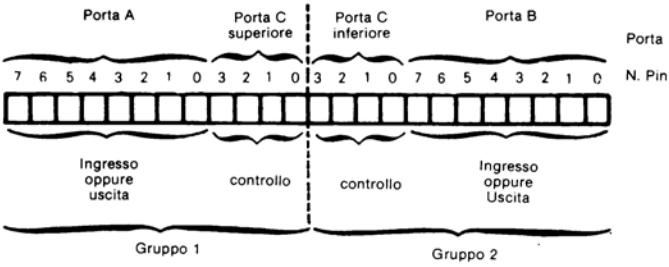
MODI DELLA PORTA I/O

Ogni gruppo di pins può essere programmato per funzionare in uno dei tre modi. I Gruppi di porte 1 e 2 non devono funzionare allo stesso modo.

Nel modo 0 ogni porta è sia una semplice porta d'ingresso che d'uscita:

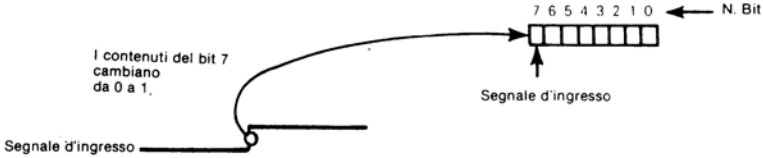


Nel modo 1, la Porta A (o B) è una porta d'ingresso con strobe oppure una porta di uscita con strobe. Lo strobe ed i segnali di controllo sono forniti dai pin della Porta C dello stesso gruppo.



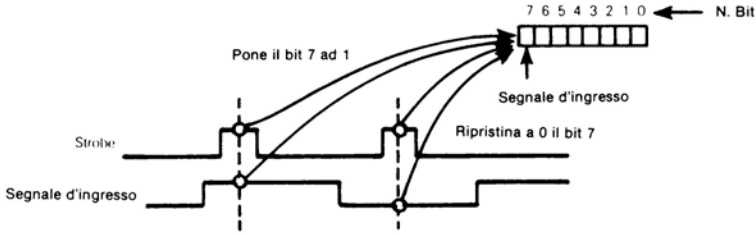
I/O SEMPLICE

Qual'è la differenza tra una porta I/O di base (Modo 0) ed una porta I/O con strobe (Modo 1)? Il modo I/O di base accetta e trasmette i dati immediatamente. Un segnale d'ingresso che sta variando stato imporrà un bit all'interno del buffer della porta I/O:

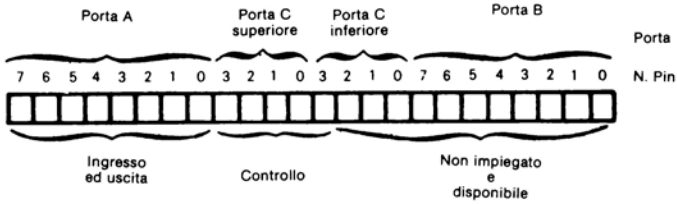


I/O CON STROBE

Una porta I/O con strobe impiega dei segnali di controllo per determinare l'istante in cui deve essere registrato un cambiamento di dati. Ancora, con riferimento ad un segnale d'ingresso, questo può essere illustrato come segue:



I pin I/O dell'8255 possono anche essere programmati per funzionare come una singola porta I/O ad 8-bit bidirezionale basata su cinque segnali di controllo:



Come illustrato precedentemente l'8255 PPI è impiegato nel Modo 2.

INDIRIZZAMENTO DELLA PORTA I/O

Indipendentemente dal modo, ogni PPI ha assegnati quattro indirizzi di porta I/O. Tre pin dell'8255 sono impiegati per selezionare il dispositivo ed una porta del dispositivo, come segue:

$\overline{CS}$: ingresso 0 per selezionare il dispositivo. Ingresso 1 per disinserirlo.

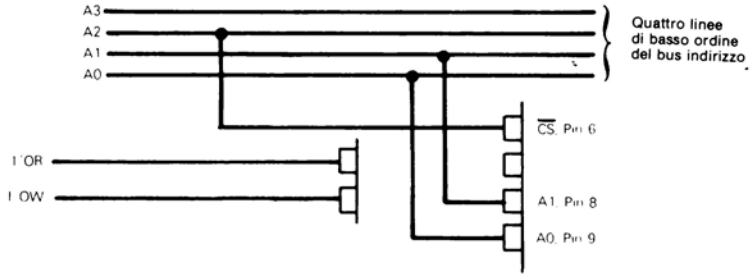
A1	A0	
0	0	Indirizzo Porta A
0	1	Indirizzo Porta B
1	0	Indirizzo Porta C (superiore od inferiore, come una singola unità ad 8 bit)
1	1	Indirizzo della "porta di controllo" del dispositivo PPI

MODO DI SELEZIONE DELLA PORTA I/O

Il dispositivo "porta di controllo" è solo una porta di scrittura. Si selezionano i modi della porta scrivendo un codice opportuno nella porta di controllo. Una discussione dettagliata dei codici della porta di controllo non aiuterà a capire la materia trattata in questo capitolo, quindi si rimanda per tale discussione ad An Introduction To Microcomputers - Volume II - Some Real Products.

DETERMINAZIONE DELL'INDIRIZZO DELLA PORTA I/O

Ora quando viene eseguita un'istruzione IN od OUT da una CPU tipo 8080, il numero della porta esce sulle 8 linee del bus indirizzo di basso ordine. Perciò si connetterà l'8255 PPI come segue:



Queste connessioni di pin forzeranno come segue le porte I/O dell'8255:

	A2	A1	A0
Porta 0-PPI Porta A	0	0	0
Porta 1-PPI Porta B	0	0	1
Porta 2-PPI Porta C	0	1	0
Porta 3-PPI Porta di Controllo	0	1	1

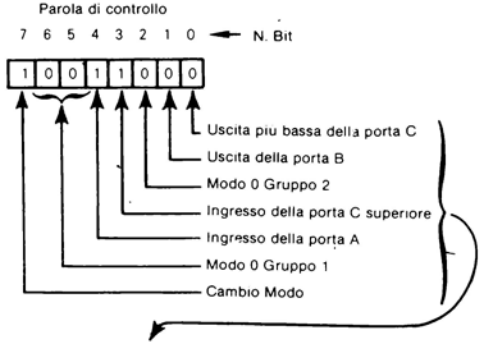
DECODIFICA DEL BUS INDIRIZZI

Si osservi che si deve decodificare con cura il bus indirizzo quando si stanno impiegando dispositivi a semplice I/O come la porta I/O 8212. Il bus indirizzo dei dispositivi microprocessori tipo 8080 fa uscire l'informazione diagnostica quando il bus indirizzo è presumibilmente inattivo. Operando l'AND del chip selezionato con appropriati segnali di Controllo del Bus, in questo caso $\overline{I/OR}$ OR $\overline{I/OW}$, si assicura che il bus indirizzo è decodificato solo quando deve essere iniziata una operazione e I/O.

Inizialmente, per semplicità, si programmerà l'8255 PPI per funzionare nel Modo 0, con assegnate all'ingresso le porte del Gruppo 1 ed all'uscita quelle del Gruppo 2.

SEQUENZA DI ISTRUZIONE PER LA SELEZIONE DEL MODO DELLA PORTA I/O

Per comprendere la presente discussione non è necessario conoscere come l'8255 è programmato per queste richieste; tuttavia ecco una spiegazione di un'opportuna sequenza d'istruzione e controllo di parola:



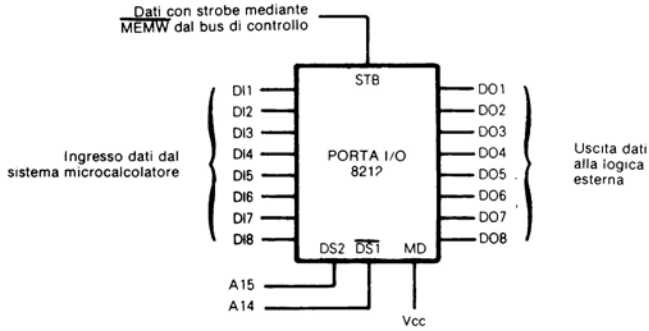

```
MVI     A,98H     ;CARICA IL BYTE DI CONTROLLO PPI
OUT     3         ;USCITA DELLA PORTA DI CONTROLLO PPI
```

Per verificare il precedente formato della parola di controllo si confronti questa con la descrizione dell'8255 PPI riportata in An Introduction To Microcomputers - Volume II - Some Real Products.

LA PORTA INGRESSO/USCITA AD 8-BIT 8212

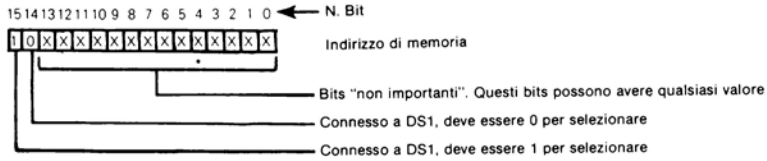
Questo dispositivo è semplicemente un buffer ingresso/uscita ad 8-bit con strobe.

La porta I/O 8212 ha 8 pin "dati in" ed 8 pin "dati out". Il modo della porta I/O è controllato dal segnale MD ed i trasferimenti dei dati sono regolati dallo strobe del segnale STB. Si impiegherà la porta I/O 8212 come magazzino dati e dispositivo d'uscita, cablato come segue:



PORTE I/O INDIRIZZATE COME MEMORIA

La porta I/O 8212 è stata cablata per auto selezionarsi in risposta alle istruzioni di reference della memoria, imponendo che i due bit di ordine elevato dell'indirizzo di memoria siano 10. In altre parole qualunque indirizzo di memoria nel range da 8000H a BFFF selezionerà la porta I/O 8212:



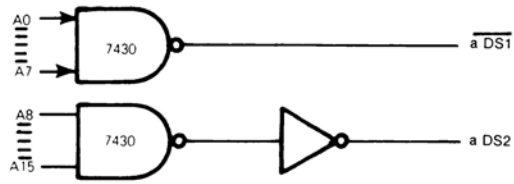
Se si osserva il range di valori, i bit "non importa" possono essere tali che, se si prende il range di indirizzi di memoria, l'8212 così cablato, risponderà:

Minimo indirizzo $\underbrace{1000}_{8}\underbrace{0000}_{0}\underbrace{0000}_{0}\underbrace{0000}_{0}$

Massimo indirizzo $\underbrace{1011}_{B}\underbrace{1111}_{F}\underbrace{1111}_{F}\underbrace{1111}_{F}$

Sono stati impiegati più di $3FFF_{16}$ indirizzi di memoria attraverso una porta I/O singola. E' importante? Non realmente. Sono stati lasciati gli indirizzi $C000_{16}$ — più lontani del necessario.

Si potrebbe riservare appena un'indirizzo di memoria per la porta I/O ma questo significherebbe sintetizzare i due segnali selezionati fuori dalle 16 linee di indirizzo. Ecco come un singolo indirizzo $FFFF_{16}$ selezionerebbe la porta I/O 8212:



Un 7430 è un gate NAND a 8-ingressi.

Si ricordi che le uscite del 7430 devono subire l'AND con segnali di controllo appropriati per assicurare che il bus indirizzo sia decodificato solo quando deve uscire un indirizzo valido.

Ma perchè sciupare denaro in logica ulteriore non necessaria?

SEGNALI D'INGRESSO

Si porrà l'attenzione ai segnali d'ingresso che appaiono nella parte sinistra della Figura 3-1. Si descriverà ogni segnale, assegnadogli un appropriato pin d'ingresso ed aggiungendo una rudimentale sequenza d'istruzione per accedere al segnale al livello più elementare.

$\overline{\text{RETURN STROBE}}$

Se l'operatore vuol vedere gli ultimi caratteri stampati, si verificano due cose:

1) **La ruota di stampa deve essere mossa alla sua posizione di visibilità.**
2) **Il nastro deve essere fatto scendere.**

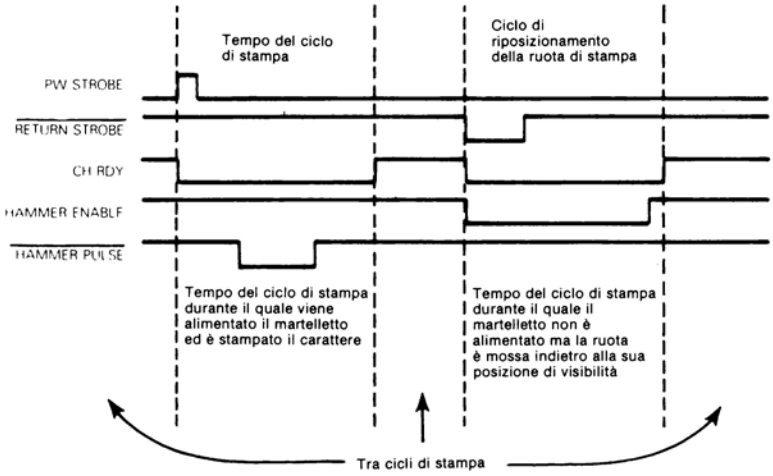
La logica esterna può occuparsi di far scendere e salire il nastro, ma la logica della Figura 3-1 origina i segnali che consentono il movimento della ruota di stampa.

Per muovere la ruota alla sua posizione di visibilità, perciò, la logica di controllo esterno del nastro fa entrare al livello basso $\overline{\text{RETURN STROBE}}$ mentre il nastro è fatto scendere.

RIPOSIZIONAMENTO DELLA RUOTA PER IL CICLO DI STAMPA

La logica compresa nella Figura 3-1 impiega $\overline{\text{RETURN STROBE}}$ come un segnale alternativo per l'inizio del ciclo di stampa; comunque $\overline{\text{RETURN STROBE}}$ basso è accompagnato da HAMMER ENABLE FF basso che previene l'alimentazione del martelletto. **Perciò un ciclo di stampa fatto partire da $\overline{\text{RETURN STROBE}}$ basso è uno pseudo-ciclo di stam-**

pa che riporta la ruota alla sua posizione di visibilità ma non alimenta il martelletto; si farà riferimento a questo come ad un ciclo di riposizionamento della ruota:



Si assegnerà il pin 4 della Porta I/O 2 a $\overline{\text{RETURN STROBE}}$.

Tra i cicli di stampa si può verificare questo pin per far partire un nuovo ciclo di stampa attraverso la seguente sequenza d'istruzione:

```
LOOP IN    2      ; Ingresso dei contenuti della porta I/O 2 all'Accumulatore
     ANI   10H    ; Maschera tutti i bit tranne il 4
     JNZ   LOOP   ; Se questo bit è 1, ritorno e ri-verifica
; La nuova sequenza d'istruzione del ciclo di stampa cominciia qui:
```

Ecco un controllo di come lavora l'istruzione ANI nella precedente sequenza:

```
7 6 5 4 3 2 1 0  ← N. Bit
X X X Y X X X X  Contenuti Accumulatore
0 0 0 1 0 0 0 0  10H
0 0 0 Y 0 0 0 0  AND
      ↑
      └──────── Questo bit corrisponde a RETURN STROBE
```

PFL REL

Il martelletto non può essere alimentato mentre il meccanismo di alimentazione della carta è in movimento perciò in questo istante gli ingressi PFL REL della logica esterna sono bassi.

La logica contenuta nella Figura 3 ritarderà l'alimentazione del martelletto finchè PFL REL permane basso.

Si assegnerà il Pin 0 della Porta I/O 0 a PFL REL.

Prima dell'esecuzione della sequenza d'istruzione che alimenta il martelletto, entreranno i contenuti della Porta 0 ed il bit di prova 0; finchè quest'ultimo bit è zero non si eseguirà la sequenza di alimentazione del martelletto.

Le istruzioni seguenti eseguono il test richiesto:

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     ANI    1      ; Maschera tutti i bit tranne il bit 0
     JZ     LOOP   ; Se il bit è 0 non alimenta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto comincia qui.
```

RIB LIFT RDY

Questo segnale è simile a PFL REL; esso **entra basso quando la logica di sollevamento del nastro sta muovendo il nastro stesso.** Esattamente come il martelletto non può essere alimentato mentre è attivo il meccanismo di alimentazione della carta così esso non può essere alimentato mentre il nastro è in movimento. Connettendo RIB RDY al Pin 1 della Porta I/O 0 si può posizionare l'inizio della sequenza d'istruzione di alimentazione del martelletto come segue:

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     CMA            ; Complementa il risultato di prova per qualsiasi bit 0 presente
     ANI    3      ; Maschera tutti i bit tranne 0 ed 1
     JNZ    LOOP   ; Qualsiasi bit 0 ora sarà 1. Se ora tutti i bit sono 1 non ali-
                   ; menta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto comincia qui.
```

PW STROBE

Questo segnale è già stato incontrato; esso è posizionato alto dalla logica esterna per iniziare un normale ciclo di stampa durante il quale sarà stampato un carattere.

Si ricordi che $\overline{\text{RETURN STROBE}}$ è basso per iniziare un ciclo di stampa durante il quale la ruota sarà mossa alla sua posizione di visibilità ma nessun carattere sarà stampato.

Assumendo che **PW STROBE sia connesso al pin 5 della Porta I/O 2,** la sequenza eseguita tra i cicli di stampa sarà la seguente:

```
LOOP IN     2      ; Ingressi dei contenuti della Porta I/O 2 all'Accumulatore
     ANI    30H    ; Isola il bit 5 (PW STROBE) ed il bit 4 (RETURN STROBE)
     CPI    10H    ; Prova per PW STROBE = 0, RETURN STROBE = 1
     JZ     LOOP   ; Se il test è verificato permane nel ciclo
; La sequenza d'istruzione del ciclo di stampa comincia qui
```

Si osservi che PW STROBE = 0 oppure $\overline{\text{RETURN STROBE}}$ = 0 devono far scattare l'inizio del ciclo di stampa; questo perchè solo PW STROBE = 0 e $\overline{\text{RETURN STROBE}}$ = 1 fanno permanere nel ciclo di istruzione di prova.

LARGHEZZA DELL'IMPULSO DEL SEGNALE DI INGRESSO

Ora le quattro istruzioni precedenti impiegano in totale per la loro esecuzione 34 cicli di clock. Con un clock di 500 nanosecondi le quattro istruzioni saranno eseguite in 17 microsecondi — che diventa la minima larghezza dell'impulso consentito per PW STROBE. **Se PW STROBE è mantenuto alto per meno di 17 microsecondi il ciclo d'istruzione può non venire eseguito.**

$\overline{\text{FFA}}$

Questo è un altro **segnale** di preavviso del martelletto. Esso **è posto a 0 mentre la logica esterna sta avanzando il nastro. Connettendo questo segnale al pin 2 della Porta I/O 0, si può modificare la sequenza d'istruzione che precede l'alimentazione del martelletto come segue:**

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     CMA            ; Complementa il risultato di prova per qualsiasi bit 0
     ANI    7      ; Isola i bit 2, 1 e 0
     JNZ    LOOP   ; Qualsiasi bit 0 ora sarà 1, se qualunque bit 1 non alimen-
                   ; ta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto inizia qui.
```

Non si è fatto altro che aggiungere una condizione di prova in più che deve essere verificata prima dell'esecuzione della sequenza d'istruzione dell'alimentazione del martelletto.

RESET

Questo è un segnale incontrato comunemente nei più svariati tipi di logica. Esso è **un segnale di inizializzazione.** Il suo scopo è di assicurare che tutta la logica sia in uno stato iniziale che, nel caso in questione, è la condizione esistente tra cicli stampa.

La logica della Figura 3-1 connette il segnale RESET ai dispositivi logici tali che RESET, passando al livello alto, forzi tutta la logica ad una condizione iniziale.

RESET DELLA CPU

Ci sono molti modi in cui un sistema a microcalcolatore può manipolare un segnale RESET. Lo schema più semplice è di fare uscire questo segnale dal pin RESET della CPU 8080.

Un altro metodo di manipolazione di RESET è di provare il segnale tra cicli di stampa e prevenire qualsiasi inizio di ciclo di stampa iniziato mentre RESET è alto; questo può essere accompagnato dalla connessione di RESET al pin 6 della Porta I/O 2, quindi modificando la sequenza di istruzione "tra i cicli di stampa" come segue:

```
LOOP IN     2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     ANI    40H    ; Isola il bit 6 (RESET)
     JNZ    LOOP   ; Se RESET è alto permane nel ciclo
; Se RESET è basso si verifica PW STROBE e RETURN STROBE
     IN     2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     ANI    30H    ; Isola il bit 5 (PW STROBE) ed il bit 4 (RETURN STROBE)
     CPI    10H    ; Verifica per PW STROBE = 0, RETURN STROBE = 1
     JZ     LOOP   ; Se verifica soddisfatta permane nel ciclo
; La sequenza di istruzione del ciclo di stampa comincia qui
```

LARGHEZZA DELL'IMPULSO DEL SEGNALE

Ora questo ciclo di prova più lungo per essere eseguito richiederà 61 cicli. Questo implica che PW STROBE deve essere mantenuto alto almeno 30,5 microsecondi, assumendo un clock di 500 nanosecondi.

PFR REL

Questo è un altro **segnale** che deve essere verificato prima dell'alimentazione del martelletto. Esso **indica quando la logica esterna sta muovendo l'alimentazione della carta.** In queste condizioni non si può alimentare il martelletto. **Connettendo questo segnale al pin 3 della Porta d'ingresso 0 si deve soltanto posizionare la sequenza di inizializzazione per l'istruzione di alimentazione del martelletto come segue:**

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     CMA            ; Complementa il risultato per provare qualsiasi bit 0
     ANI    0FH    ; Isola i bit 3, 2, 1 e 0
     JNZ    LOOP   ; Qualsiasi bit 0 ora sarà 1, se qualsiasi bit è ora 1, non ali-
                   ; menta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto comincia qui
```

CA REL

Questo segnale è pressocchè identico a PFR REL. Esso **proviene dalla logica esterna che controlla il movimento del carrello. Si connetterà questo segnale al pin 4 della Porta di ingresso 0** e si modificherà la sequenza d'inizializzazione per l'istruzione di

alimentazione del martelletto come segue:

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     CMA           ; Complementa il risultato per provare qualsiasi bit 0
     ANI    1FH    ; Isola i bit 4, 3, 2, 1 e 0
     JNZ    LOOP   ; Qualsiasi bit 0 ora sarà 1, se qualsiasi bit è ora 1 non ali-
                   ; menta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto comincia qui
```

FFI

Questo è il segnale che regola il primo ritardo del ciclo di stampa — cioè il tempo durante il quale la ruota si muove dalla sua posizione di visibilità finchè il petalo richiesto è di fronte al martelletto.

FFI è generato dalla logica esterna, esso è basso mentre la ruota è in movimento ed alto nel caso contrario.

RITARDO DI TEMPO BASATO SUL SEGNALE D'INGRESSO

Si connetterà FFI al pin 7 della Porta I/O 0. Il ciclo di istruzione seguente originerà un ritardo che dura finchè FFI diventa alto:

```
LOOP IN     0      ; Ingresso dalla Porta 0 all'Accumulatore
     RLC           ; Scorre il bit 7 nel Carry
     JNC    LOOP   ; Se Carry = 0 permane nel ciclo
```

Si vede come opera questo ciclo? Dopo che i contenuti della Porta I/O 0 sono entrati nell'Accumulatore si è interessati solo al bit 7 poichè questo è il bit che corrisponde ad FFI.

Questo è quanto esegue l'istruzione RLC:

Se lo stato Carry è uguale ad 1 termina il ritardo del movimento della ruota. Se Carry è uguale a 0 la logica del programma deve continuare il ritardo.

$\overline{\text{EOR DET}}$

Questo segnale indica che la fine del nastro è stata raggiunta. In queste condizioni la stampa del carattere non può continuare.

Quando viene generato il segnale, ci sarà ancora del nastro fresco davanti al martelletto cosicchè il segnale non è impiegato per inibire l'alimentazione del martelletto; invece esso e impiegato per prevenire la fine del ciclo di stampa. Effettivamente questo previene l'inizio di un nuovo ciclo di stampa.

Si connetterà il segnale $\overline{\text{EOR DET}}$ al bit 7 della Porta I/O 2. Poichè $\overline{\text{EOR DET}}$ è un segnale in logica negativa esso sarà verificato prima di entrare nella fase "tra cicli di stampa" come segue:

```
; Verifica della validità della fine del ciclo di stampa
LOP1 IN     2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     RLC           ; Scorre il bit 7 nel Carry
     JNC    LOP1   ; Se il Carry è zero, permane nel ciclo di stampa
```

```
; Inizio della fase tra cicli di stampa
LOOP IN      2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     ANI     40H    ; Isola il bit 6 (RESET)
     JNZ     LOOP   ; Se RESET è alto permane in questa fase
; RESET è basso. Verificare PW STROBE e RETURN STROBE
     IN      2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     CPI     10H    ; Verifica se PW STROBE = 0, RETURN STROBE = 1
     JZ      LOOP   ; Se la verifica è soddisfatta permane nel ciclo
; La sequenza d'istruzione del ciclo di stampa inizia qui
```

Si osservi la precedente sequenza d'istruzione. In essa ci sono alcuni aspetti interessanti.

Le prime tre istruzioni precedenti saranno le ultime tre istruzioni della sequenza del ciclo di stampa. L'istruzione con la label LOOP è la prima istruzione della sequenza che viene eseguita continuamente finchè non inizia il successivo ciclo di stampa. Così se $\overline{\text{EOR DET}}$ e basso la logica del programma si aggancerà alle prime tre istruzioni precedenti, ciclando continuamente all'interno di queste tre istruzioni finchè $\overline{\text{EOR DET}}$ diventa alto. In questo istante il ciclo di stampa termina e si esegue il ciclo di istruzione "tra il ciclo di stampa". Il programma ora si aggancia indefinitivamente a questo anello d'istruzione finchè il bit 6, corrispondente a RESET è uguale a 0, mentre il bit 5, corrispondente a PW STROBE, è uguale ad 1 oppure il bit 4, corrispondente a RETURN STROBE è uguale a 0.

C'è un'altra caratteristica interessante nella precedente sequenza d'istruzione. Si potrebbe, volendo, eliminare la seconda istruzione IN, come segue:

```
; Verifica per termine valido del ciclo di stampa
LOP1 IN      2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     RLC             ; Sposta il bit 7 nel Carry
     JNC     LOP1   ; Se il Carry è zero, permane nel ciclo di stampa
; Inizio dell'anello tra cicli di stampa
     ANI     80H    ; Isola il bit 6 (RESET)
     JNZ     LOP1   ; Se RESET è alto permane nell'anello
; RESET è basso. Verificare PW STROBE e RETURN STROBE
     IN      2      ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
     ANI     30H    ; Isola il bit 5 (PW STROBE) ed il bit 4 (RETUNR STROBE)
     CPI     10H    ; Verifica se PW STROBE = 0, RETURN STROBE = 1
     JZ      LOP1   ; Se la verifica è soddisfatta permane nel ciclo
; La sequenza d'istruzione del ciclo di stampa comincia qui
```

Eliminando un'istruzione sono stati risparmiati due byte del codice oggetto. La penalità consiste nel fatto che sono stati aggiunti 14 cicli di clock all'intero ciclo d'istruzione, questo significa che l'impulso alto in PW STROBE supera i 30,5 microsecondi calcolati durante la discussione del segnale RESET portandosi a 37,5 microsecondi.

Perchè funziona la sequenza d'istruzione condensata, sopra illustrata? La ragione è perchè la logica esterna non è supposta che muova il nastro tra i cicli di stampa perciò $\overline{\text{EOR DET}}$ sarà sempre alto durante l'anello di esecuzione dell'istruzione tra i cicli di stampa. Se è così l'istruzione RCL sposterà sempre un 1 nel Carry che imporrà sempre all'esecuzione di continuare con l'istruzione ANI. Così le prime tre istruzioni diventano innocue. Si noti che l'istruzione ANI 40 è diventata un'istruzione ANI 80 poichè il bit del segnale RESET è stato spostato di una posizione di sinistra dall'istruzione RLC.

HAMMER ENABLE FF

Questo segnale previene che il martelletto venga alimentato dopo che la ruota è mossa alla sua posizione di visibilità come descritto in riferimento al segnale $\overline{\text{RETURN STROBE}}$.

Si connetterà $\overline{\text{HAMMER ENABLE FF}}$ al pin 6 della Porta I/O 0, quindi si modificherà la sequenza d'istruzione, che precede l'alimentazione del martelletto, come segue:

```
LOOP IN     0      ; Ingresso dei contenuti della Porta I/O all'Accumulatore
     ANI    5FH    ; Isola i bit 6, 4, 3, 2, 1 e 0
     CMA           ; Complementa il risultato per verificare qualsiasi bit 0
     JNZ    LOOP   ; Qualsiasi bit 0 ora sarà 1, se qualunque bit è 1, non ali-
                   ; menta il martelletto
; La sequenza d'istruzione per l'alimentazione del martelletto comincia qui
```

CLK

Questo è il segnale di clock che sincronizza tutta la logica della Figura 3–1. Si provi a vedere come non sia possible includere questo segnale nella simulazione di Figura 3–1, poichè gli eventi all'interno del programma del microcalcolatore devono essere sincronizzati dalla sequenza in cui sono sincronizzate le istruzioni — e non mediante un clock. **Analogamente i successivi due segnali; +5 ed RV1 sono alimentazioni. Essi non hanno significato all'interno di un programma del microcalcolatore.**

H1 — H6

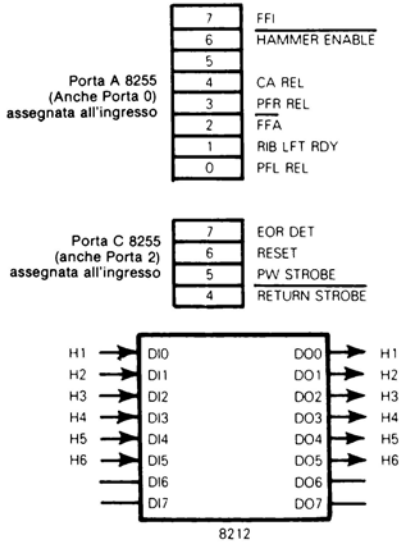
Questi sono sei segnali che selezionano uno dei sei intervalli di tempo di durata per l'impulso di alimentazione del martelletto. Si assegneranno questi segnali alla porta I/O 8212. Una volta eseguita la sequenza di alimentazione del martelletto, questi segnali vengono semplicemente caricati nell'Accumulatore, come segue:

```
LDA     H1, H6  ; Ingresso del codice del tempo impulso di alimentazione
```

H1 H6 è la label a quattro caratteri rappresentante l'indirizzo di memoria della Porta I/O "selezionata".

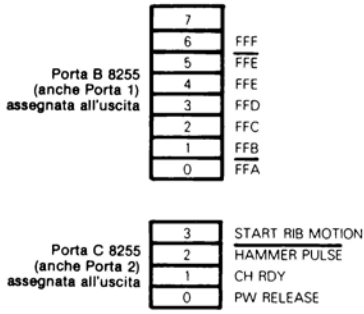
SOMMARIO SUL SEGNALE D'INGRESSO

In conclusione questo è quanto assegnato a proposito dei segnali d'ingresso:



SEGNALI D'USCITA

Si porrà ora l'attenzione sui segnali d'uscita elencati sulla parte destra della Figura 3-1. Questi segnali sono molto più facili da descrivere dei segnali d'ingresso. Essi consistono di sei uscite flip-flop — che sono semplicemente indicatori di timing impiegati dalla logica esterna — più quattro segnali di controllo. **Questi segnali provengono dalle porte più basse B e C dell'8255 PPI come segue:**



Si assegnerà un pin per FFC anche se non è un'uscita perchè la Porta I/O 1 serve ad un doppio scopo — come locazione di magazzino dati e come un buffer dei segnali di uscita. Le routine semplici per generare segnali d'uscita non possono essere mescolate; questo è l'intero scopo della logica di Figura 3-1. Quindi si definiranno semplicemente i quattro segnali di controllo:

1) **PW REL — Questo segnale contrassegna la fine del ritorno del martelletto, il posizionamento del ritardo di tempo e l'inizio del ritardo fisso del Movimento Finale durante** il quale la logica esterna può muovere l'alimentazione della carta ed il carrello.

2) **CH RDY — Questo è anche indicato segnale di MARTELLETTO PRONTO. Questo è il segnale che definisce l'intero intervallo di tempo del ciclo di stampa;** esso diventa basso all'inizio del ciclo di stampa e permane basso fino alla fine dello stesso.

3) **IMPULSO DEL MARTELLETTO. Questo segnale deve essere basso per l'intervallo di tempo durante il quale la logica esterna trasmette un impulso di alimentazione del solenoide del martelletto.**

4) **IMPULSO DI INIZIO MOVIMENTO DEL NASTRO. Questo segnale è mantenuto alto prima del ciclo di stampa per informare la logica esterna della sicurezza di avanzamento del nastro in modo che venga a trovarsi del nastro fresco davanti al martelletto quando questo è premuto.**

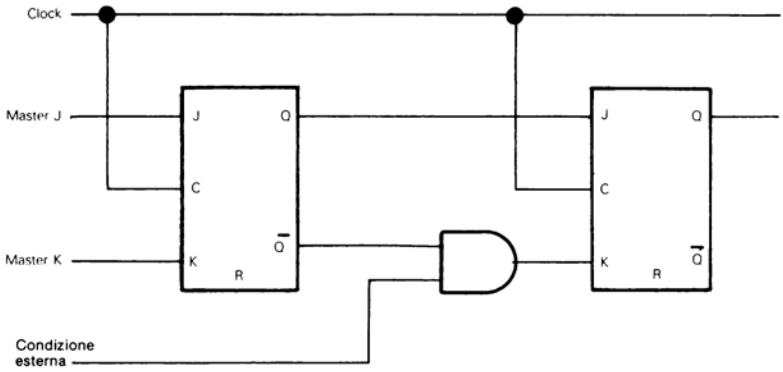
UNA SIMULAZIONE ORIENTATA ALLA LOGICA DIGITALE

Si è già pronti per iniziare la simulazione della logica illustrata in Figura 3-1 — ma prima si farà una breve analisi della logica.

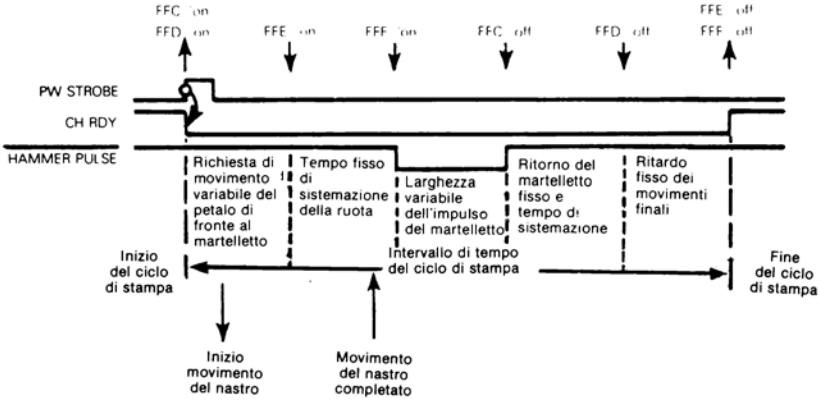
UN'ANALISI DELLA LOGICA

Al centro della sequenza logica ci sono quattro flip-flop 74107, indicati FFC_W, FFD_W, FFF_W. Si troveranno questi flip-flop al centro ed a sinistra della Figura 3-1. **Questi**

quattro flip-flop formano il cosidetto "Contatore Johnson". Ogni flip-flop è controllato dall'uscita di quello precedente accoppiato con un test per le condizioni esterne:



Così i quattro flip-flop possono essere visualizzati come eventi di inizializzazione del ciclo di stampa come segue:



Come illustrato sopra, l'intervallo di tempo del ciclo di stampa può essere diviso in cinque periodi.

Durante il primo intervallo di tempo la ruota di stampa è mossa dalla sua posizione di visibilità finchè il petalo richiesto è di fronte al martelletto. Questo intervallo di tempo è controllato dalla logica esterna, attraverso l'ingresso FFI.

I quattro intervalli di tempo rimanenti sono controllati dai tre one-shots 74121 ed il multivibratore 555.

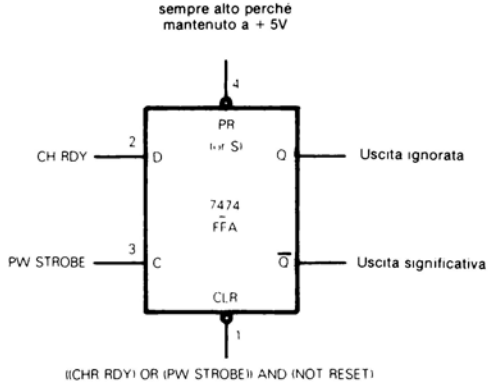
Invece qual'è lo scopo dei due flip-flop all'estremità dell'angolo sinistro della Figura 3-1? Questi costituiscono semplicemente la logica di inizializzazione del ciclo. Il flip-flop FFA è comandato da una combinazione di segnali necessari per iniziare un ciclo di stampa. Il flip-flop FFB agisce come interruttore per i quattro flip-flop 74107 forzandoli allo spegnimento tra i cicli di stampa. Il flip-flop FFB fa questo collegando la sua uscita Q all'ingresso reset dei flip-flop 74107. Questo fa sì che i flip-flop 74107 siano sempre spenti se lo è FFB; in seguito si spiegherà più dettagliatamente come questo avviene.

Si vuole ora seguire un ciclo di stampa attraverso la Figura 3-1. Contemporaneamente si costruirà il programma in linguaggio assembly del microcalcolatore che simula la logica, dispositivo per dispositivo.

FLIP-FLOP FFA_W

FLIP-FLOP 7474

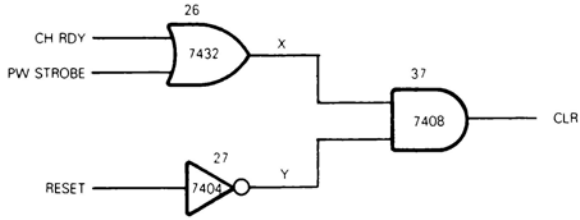
Il ciclo di stampa comincia da un flip-flop 7474 indicato FFA_W. Si troverà questo flip-flop all'estremo angolo in alto nella Figura 3-1. Si isoli FFA_W e lo si illustri come segue:



Ci si riferisca alla tabella generale di funzione per un flip-flop 7474, fornita al Capitolo 2.

Poichè PRESET (PR) è sempre alto, essendo collegato a +5V, un ingresso CLEAR (CLR) basso forzerà lo spegnimento del flip-flop all'istante in cui Q è basso e $\overline{Q}$ alto.

Si osservi la Figura 3-1. e si vedrà che CLR è generato in questo modo:



Questa è la tabella della verità per CLR:

CH RDY	PW STROBE	X	RESET	Y	CLR
0	0	0	0	1	0
			1	0	0
0	1	1	0	1	1
			1	0	0
1	0	1	0	1	1
			1	0	0
1	1	1	0	1	1
			1	0	0

Perchè il flip-flop FFA_W si spenga, CLR deve essere alto; perchè CRL sia alto, RESET deve essere basso e CH RDY, oppure PW STROBE, deve essere alto.

Ora CH RDY fornisce FFA_W con i suoi dati (D) d'ingresso e PW STROBE fornisce l'ingresso del clock (C). Perciò la tabella di funzione per il flip-flop FFA_W deve essere illustrata come segue:

INGRESSI				USCITE		
PRESET	CLR	CLOCK (PW STROBE)	D (CH RDY)	Q	$\bar{Q}$	
0	1	0 o 1	0 o 1	1	0	PRESET=1
1	0	0 o 1	0 o 1	0	1	
0	0	0 o 1	0 o 1	Instabile		PRESET=1
1	1	0 → 1	1	1	0	
1	1	0 → 1	0	0	1	
1	1	0	0 o 1	Precedente Q	Precedente Q	Nessun cambiamento

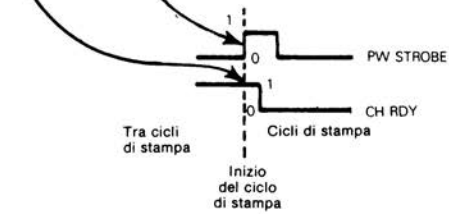
E questo porta alla seguente piccola tabella di funzione:

CLR	CH RDY	PW STROBE	$\bar{Q}$	
0			1	condizione "off"
1	0	0 → 1	1	condizioni "on" possibili
1	1	0 → 1	0	condizioni "on" possibili

Si assume che una transizione da 0 ad 1 del PW STROBE per il flip-flop FFA_W lo spenga. Comunque quando FFA_W si spegne, se CH RDY è zero, allora "uscita $\bar{Q}$ è ancora 1", rappresentando la condizione di spegnimento. Così per accendere FFA_W, PW STROBE deve andare da 0 ad 1 mentre CH RDY è 1.

Si ricordi che CH RDY è un segnale alto tra i cicli di stampa ed è basso durante un ciclo di stampa. Questo significa che il flip-flop FFA_W sarà acceso solo se PW STROBE è alto tra i cicli di stampa caratterizzati da CH RDY alto:

CLR	CH RDY	PW STROBE	$\bar{Q}$
0			1
1	0	0→1	1
1	1	0→1	0



Per il momento non ci si preoccupi di come CH RDY vada brevemente a 0 dopo lo spegnimento del FFA_W. Successivamente si spiegherà come questo avviene. La sola cosa importante da notare è che un impulso alto di PW STROBE sarà ignorato se si verifica mentre CH RDY è basso.

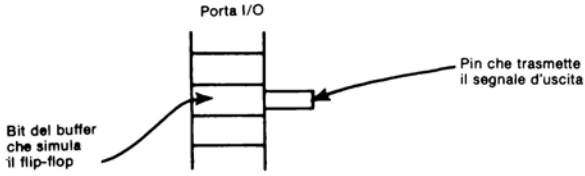
RESET **E per quanto riguarda il segnale RESET? Questo segnale supera tutta la logica associata col flip-flop FFA_W; ogni volta che RESET è alto, CLR è forzato basso spegnendo incondizionatamente FFA_W ogni volta che è acceso.**

SIMULAZIONE DEL FLIP-FLOP FFA_W

E' stato concluso al Capitolo 2 che un flip-flop, in un sistema a microcalcolatore, è rappresentato da un singolo bit della memoria di lettura/scrittura. Un singolo bit di un buffer di lettura/scrittura rappresenta la stessa cosa.

SIMULAZIONE DEL FLIP-FLOP IMPIEGANDO PORTE I/O

La Porta I/O 1 è stata assegnata ai segnali d'uscita. Questa porta ha un buffer ad 8-bit al quale sono connessi i pin della porta; così **ogni bit del buffer della porta simulerà il flip-flop le cui uscite sono trasmesse attraverso il pin della porta:**



Si ricordi che FFA è stato assegnato al pin 0 della Porta I/O 1.

Ora si è pronti per simulare il flip-flop FFA_W.

Allo stesso tempo come simulare i tre gate sotto ed a sinistra di FFA_W? Questi tre gates sono numerati 26, 27 e 37 ed insieme originano l'ingresso CLR.

Per la simulazione individuale di questi tre gates si applichi la seguente sequenza di istruzione:

```
; Simula il gate 27
        IN      2       ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
        CMP             ; Complementa tutti gli otto bit
        ANI     40H     ; Isola il bit 6; esso rappresenta il complemento di RESET
        MOV     B,A     ; Conserva il complemento nel registro B
; Simula il gate 26
        IN      2       ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
        ANI     22H     ; Si isolano i bit 5 ed 1, essi rappresentano
        MOV     C,A     ; PW STROBE e CH RDY, conserva nel registro C della CPU
; Simula il gate 37
        JZ      CLR0    ; Se il bit 1 od il bit 5 non è uguale ad 1, CLR è 0
        MOV     A,B     ; Prova il complemento di RESET mediante movimento
        AND     A       ; All'Accumulatore ed AND con sé stesso
        JZ      CLR0    ; Se il risultato è 0, CLR è 0
        MVI     D,1     ; CLR è 1 così immagazzina 1 IND D, bit 0
        JMP     FFAW
CLR0    MVI     D,0     ; CLR è 0 così immagazzina 0 in D, bit 0
; Simula il flip-flop FFAW
FFAW    MOV     A,D     ; Prova lo stato di CLR muovendo D al-
                RRC     ; l'Accumulatore e facendo scorrere il bit 0 nel Carry
        JNC     FFA0    ; Se CLR è 0, pone il bit 0 della Porta I/O 1 a 1
        MOV     A,C     ; CLR non è zero, prova PW STROBE
        ANI     20H     ; Se PW STROBE è zero, il clock non è stato pulsato
        JZ      FFA0    ; Pone il bit 0 della Porta I/O 1 ad 1
        MOV     A,C     ; PW STOBE è 1, prova CH RDY
```

```
      ANI    02H    ; Se CH RDY è 0, pone il bit 0 della Porta I/O 1 a 0
      JZ     FFA0   ;
      IN     1      ; Carica la Porta I/O 1 nell'Accumulatore
      ANI    F7H    ; Il bit 0 deve essere ripristinato a 0, poichè FFA sia acceso
      OUT    1      ;
      JMP    FFB    ; Salta alla simulazione del flip-flop B
FFA0  IN     1      ; Carica la Porta I/O 1 nell'Accumulatore
      ORI    1      ; Il bit 0 deve essere posto ad 1, poichè FFA sia spento
      OUT    1      ;
, Segue la simulazione del flip-flop FFB
```

E' molto importante comprendere come le istruzioni si adattano insieme per formare un programma. Non si prosegua se non si è capito a fondo come la precedente sequenza simula la logica dell'FFA_W con i tre gates associati.

Si osservino le precedenti simulazioni.

SIMULAZIONE DELL'INVERTITORE

Il segnale RESET, come si ricorderà, è stato collegato al bit 6 della Porta C I/O 8255; questa porta è indirizzata come la Porta I/O 2 basata sul modo in cui è stato fatto il collegamento dell'8255 PPI nel sistema a microcalcolatore. Per invertire questo segnale si inviano i contenuti della Porta I/O 2 all'Accumulatore, si complementano i contenuti dell'Accumulatore e quindi si isola il complemento di RESET ponendo a 0 tutti i bit dell'Accumulatore eccetto il bit 6:

		Dalla Porta I/O 2	
IN	2	XXXXXXXX	all'Accumulatore
CMP		$\overline{X}\,\overline{X}\,\overline{X}\,\overline{X}\,\overline{X}\,\overline{X}\,\overline{X}\,\overline{X}$	Complementa
ANI	40H	01000000	Isola il bit 6
		$0\overline{X}000000$	

Il complemento di RESET è conservato temporaneamente nel Registro B della CPU. **La simulazione del gate 27 è così completa.**

SIMULAZIONE DEL GATE OR

FLAGS DI STATO IMPIEGATI PER RAPPRESENTARE LA LOGICA

La simulazione del gate 26 non è altrettanto immediata. Si vuole eseguire l'OR di PW STROBE e CH RDY. Questi due segnali sono rappresentati rispettivamente dai bit 5 ed 1 della Porta I/O 2. Ora si caricano i contenuti della Porta I/O 2 nell'Accumulatore, quindi si esegue un'istruzione ANI che pone a 0 tutti i bit tranne i 5 ed 1. Ma effettivamente non si esegue l'OR di questi bit. Perchè? La ragione è che quando viene eseguita un'istruzione ANI essa pone lo stato Zero al complemento di (PW STROBE) OR (CH RDY).

	Contenuti dell'accumulatore									
A5 OR A1	A7	A6	A5	A4	A3	A2	A1	A0	Valore Esadec.	Stato Zero
0	0	0	0	0	0	0	0	0	00	1
1	0	0	0	0	0	0	1	0	02	0
1	0	0	1	0	0	0	0	0	20	0
1	0	0	1	0	0	0	1	0	22	0

PW STROBE

CH RDY

Seguendo l'esecuzione dell'istruzione ANI lo stato zero è il complemento di PW STROBE OR CH RDY

Si può perciò passare al gate 37.

STATO ZERO

Lo scopo del gate 37 è di generare l'ingresso CLR dell'FFA_W. Si simulerà CLR impiegando il bit 0 del Registro D della CPU. Ora si ha il passaggio dalla simulazione del gate 26 a quella del gate 37; in questo istante lo Stato Zero sarà 0 se l'OR di PW STROBE e CH RDY è 1; altrimenti lo stato Zero sarà 1. (Si ricordi che gli stati Zero rappresentano sempre l'inversno della condizione 0. In altre parole una condizione 0 pone lo stato Zero ad 1 e viceversa).

La prima istruzione della simulazione del gate 37 si avvale del fatto di avere l'OR di PW STROBE e CH RDY registrato nello stato zero. Se lo stato Zero è 1, CLR deve essere 0, così la prima istruzione JZ separa la logica, che porrà a 0 il bit 0 del registro D della CPU. Il gruppo successivo di istruzioni della simulazione del gate 37 prova il complemento di RESET, memorizzato nel Registro B della CPU, muovendo i contenuti del Registro B nell'Accumulatore ed eseguendo l'AND dell'Accumulatore con sé stesso. Questo AND non cambia i contenuti dell'Accumulatore ma ripristinerà gli stati in base al risultato dell'AND. Se il complemento di RESET è 0 allora l'istruzione JZ che segue separerà la logica del programma che pone a 0 il bit 0 del Registro D della CPU. Se il complemento di RESET non è 0, allora tutte le condizioni impongono al gate 37 di emettere un risultato non zero — e questa condizione è simulata dall'istruzione MVI D,1 che pone ad 1 il bit 0 del Registro D della CPU.

Di seguito viene simulato il flip-flop FFA. Lo stato di questo flip-flop può essere definito come segue:

Se CLR è 0 $\overline{Q}$ è 1.
Se PW STROBE è 0 allora $\overline{Q}$ è 1.
Se CLR è 1, PW STROBE è 1 e CH RDY è 0 allora $\overline{Q}$ è 1.
Se CLR è 1, PW STROBE è 1 e CH RDY è 1 allora $\overline{Q}$ è 0.

CLR è simulato dal bit 0 del Registro D. PW STROBE è simulato dal bit 5 del registro C. CH RDY è simulato dal bit 1 del Registro C.

La simulazione del flip-flop FFA comincia con l'istruzione avente label FFA_W.

STATO CARRY

Prima si prova lo stato di CLR muovendo i contenuti del Registro D all'Accumulatore e facendo scorrere il bit 0 nello stato Carry. Queste due fasi sono necessarie poichè le istruzioni di scorrimento dell'8080 operano solo sui contenuti dell'Accumulatore. Se il Carry è 0 allora il bit 0 del Registro D deve essere 0, questo significa che CLR è 0; perciò si seleziona l'istruzione che pone ad 1 il bit 0 della Porta I/O 1. (Si osservi che $\overline{Q}$ è simulata perchè per le condizioni "off" si pone ad 1 il bit 0 della Porta I/O 1 invece di ripristinarlo a 0).
Supponendo che CLR sia 1 successivamente si prova PW STROBE. Per fare questo si muovono i contenuti del registro C nell'Accumulatore, quindi si isola il bit PW STROBE mediante l'AND con una maschera opportuna.
Se PW STROBE è 0 si deve ancora separare l'istruzione che pone il bit 0 della Porta I/O 1 ad 1.
Assumendo che PW STROBE sia 1 rimane da controllare la condizione su CH RDY. Per fare questo si muovono ancora i contenuti del registro C nell'Accumulatore quindi si isola CH RDY operando l'AND con una maschera opportuna. Se CH RDY è 0 si separa l'istruzione che pone il bit 0 della Porta I/O 1 ad 1.

Assumendo che siano state verificate tutte le condizioni che fanno commutare un flip-flop FFA si deve porre a 0 il bit 0 della Porta I/O 1. Questo viene eseguito inviando i contenuti della Porta I/O 1 all'Accumulatore, operando l'AND con una maschera appropriata e quindi riportando il risultato:

7 6 5 4 3 2 1 0	N. Bit
X X X X X X X Y	Contenuto dell'Accumulatore
1 1 1 1 1 1 1 0	F7H
X X X X X X X 0	AND

COMMUTAZIONE ON DI UN BIT

Le ultime tre istruzioni della simulazione del flip-flop FFA pongono ad 1 il bit 0 della Porta I/O 1 (riflettendo il fatto che il flip-flop FFA è spento). Queste tre istruzioni caricano i contenuti della Porta I/O 1 nell'Accumulatore, operano l'OR con una maschera opportuna, quindi riportano il risultato:

7 6 5 4 3 2 1 0	N. Bit
X X X X X X X Y	Contenuto dell'Accumulatore
0 0 0 0 0 0 0 1	1
X X X X X X X 1	OR

Ora, in tutta onestà, la sequenza del programma appena descritta è un modo ridicolo di simulazione del flip-flop FFA e dei suoi tre gates associati.

Essa è ridicola perchè si simula ogni gate con una funzione di trasferimento indipendente. Invece si consideri il flip-flop, con i suoi tre gates, come una singola funzione di trasferimento. Si può rappresentare la funzione di trasferimento con la sequente definizione di stato:

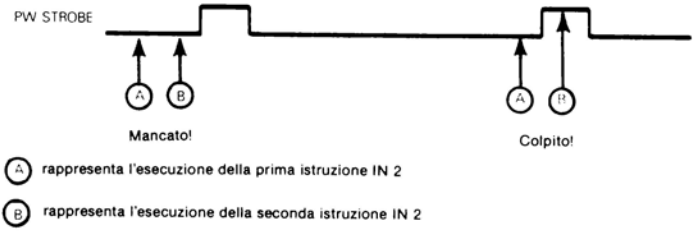
Pone $\overline{Q}$ a 0 se RESET=0, CH RDY=1 e PW STROBE va da 0 ad 1. Altrimenti pone $\overline{Q}$ ad 1.

Come eseguire la prova per la transizione di PW STROBE da 0 ad 1?

Impiegando interrupts la prova potrebbe essere molto semplice; ma non si vogliono usare interrupts fino al Capitolo 5.

RIVELAZIONE DI CAMBIAMENTI DI LIVELLO DI SEGNALE SENZA INTERRUPTS

Senza l'impiego di interrupts c'è un solo modo per verificare una transizione di PW STROBE da 0 ad 1. Si devono inviare i contenuti della Porta I/O 2 all'Accumulatore, isolare il bit 5, conservare il risultato, inviare nuovamente i contenuti della Porta I/O 2 all'Accumulatore ed isolare i bit 5, quindi confrontare i due bit per osservare se il vecchio valore è 0 e quello nuovo 1. Ma questo schema è rischioso; esso prenderà solo le transizioni di segnale che si verificano tra le due istruzioni da caricamento dei contenuti della Porta I/O 2 nell'Accumulatore.



TEMPORIZZAZIONE DEGLI EVENTI IN UN SISTEMA A MICROCALCOLATORE

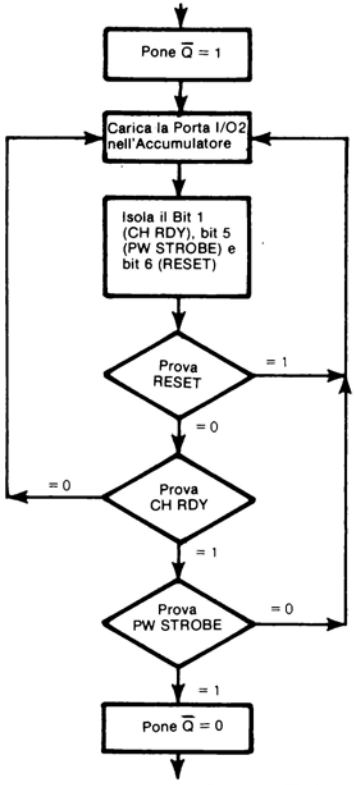
All'interno della logica di un programma al microcalcolatore, comunque, non si richiede di rivelare transizioni di segnale. Ogni sequenza è determinata dalla sequenza di esecuzione d'istruzione. L'intero concetto di timing di un gradino positivo o negativo di un impulso di segnale non ha significato. Invece di impiegare le transizioni del segnale PW STROBE verranno quindi impiegati i livelli del segnale PW STROBE. Il flip-flop FFA può ora essere descritto con la seguente definizione di stato:

Pone $\overline{Q}$ a 0 se RESET è uguale a 0, CH RDY è uguale a 1 e PW STROBE è uguale ad 1. Altrimenti pone $\overline{Q}$ ad 1.

TIMING E SEQUENZA LOGICA

Un progettista logico può essere profondamente preoccupato dalla superficiale sostituzione del trigger mediante gradino con quello mediante livello. Si può fare questo all'interno di un sistema a microcalcolatore perchè la programmazione del microcalcolatore fornisce un ulteriore grado di libertà rispetto al progetto logico digitale. L'ordine con cui si inseriscono componenti logici in una scheda di un circuito stampato non ha niente a che vedere col verificarsi di eventi logici. La sequenza logica sarà controllata dal gradino e dal livello del trigger. Ma l'ordine in cui si scrivono le istruzioni in linguaggio assembly è l'ordine in cui le istruzioni saranno eseguire.

Per guidare questo punto chiave si osservi il seguente diagramma di flusso che rappresenta la definizione di stato per il flip-flop FFA:



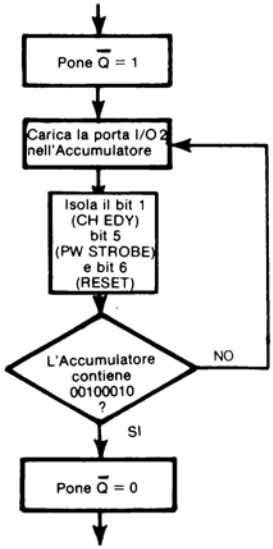
Ogni blocco rettangolare rappresenta un movimento di dati oppure un'operazione di manipolazione.

Ogni blocco di forma rombica rappresenta la logica che controlla la condizione di un flag di stato.

L'ordine in cui vengono scritte le istruzioni e quello in cui esse saranno eseguite. Con riferimento al precedente diagramma di flusso, questa sequenza di esecuzione è rappresentata dalla linea continua delle frecce rivolte verso il basso. Speciali istruzioni di Salto-Su-Condizione consentono modifiche alla normale sequenza, come rappresentato dalle frecce orizzontali che nascono dai lati dei blocchi rombici. Si può eseguire le frecce fino al punto indirizzato dal Salto-Su-Condizione.

Si riscriverà ora la simulazione del flip-flop FFA considerando il flip-flop ed i tre gates logici CLR come una singola funzione di trasferimento.

Poichè RESET, CH RDY e PW STROBE sono tutti connessi ai pin dalla Porta I/O 2, si caricano i contenuti della Porta I/O 2 nell'Accumulatore e si isolano tutti i tre bit. Ora esiste solo una combinazione di valori di questi tre bit per iniziare un nuovo ciclo di stampa. RESET deve essere uguale a 0, mentre CH RDY e PW STROBE entrambi uguali ad 1. Perciò si ridisegnerà il diagramma di flusso del programma come segue:



La sequenza d'istruzione risulta quindi condensata come segue:

```
; Simulazione dell'FFA e della logica associata
        IN      1       ; Inizialmente pone ad 1 il bit 0 della Porta I/O 1
        ORI     1
        OUT     1
; Carica i contenuti della Porta I/O 2 nell'Accumulatore
; Ed isola i bit 1, 5 e 6 per CH RDY,
; PW STROBE e RESET, rispettivamente
L10     IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ANI     62H     ; Isola i bit 6, 5 ed 1
        CPI     22H     ; Se RESET=0, CH RDY= 1 e
        JNZ     L10     ; PW STROBE=1, inizia un nuovo ciclo di stampa
        IN      1       ; Altrimenti ritorna ad L10, nuovo inizio
        ANI     FEH     ; Del ciclo di stampa ponendo a 0 il bit 0 della Porta I/O 1
        OUT     1
; La sequenza d'istruzione del nuovo ciclo di stampa inizia qui
```

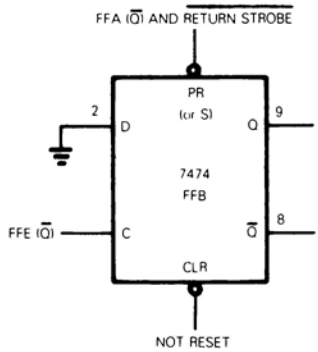
Le prime tre istruzioni della sequenza precedente pongono semplicemente ad 1 il bit 0 della Porta I/O 1. Questa è l'inizializzazione di un nuovo ciclo di stampa non incominciato. Le quattro istruzioni inizianti con la label L10 controllano la condizione che fa scattare l'inizio di un nuovo ciclo di stampa. Queste quattro istruzioni si eseguono in 34 cicli di clock che, assumendo un clock di 500 nanosecondi, significa che PW STROBE deve permanere al livello alto almeno 17 microsecondi.

Fornendo RESET uguale a 0 mentre CH RDY e PW STROBE uguale ad 1, deve iniziare un nuovo ciclo di stampa cosicchè le ultime tre istruzioni pongono a 0 il bit 0 della Porta I/O 1.

Così la simulazione del flip-flop FFA è completa.

FLIP—FLOP FFB_W

Il dispositivo successivo nella sequenza logica è un altro flip-flop 7474, indicato FFB_W nella Figura 3-1; esso si trova proprio a destra di FFA_W.
Questo flip-flop può essere illustrato come segue:



La seguente tabella di funzione descrive FFB, come collegato sopra, con il suo ingresso D collegato a 0:

FFA ($\overline{Q}$)	RETURN STROBE	PRESET	NOT RESET (CLR)	FFE ($\overline{Q}$) =CLOCK	Q	$\overline{Q}$
0	0	0	1	X	1	0
0	1	0	0	X	instabile	
1	0	0				
1	1	1	0	X	0	1
		1	1	0→1	0	1

Il Capitolo 2 fornisce la tabella di funzione del flip-flop standard 7474; qui è stata semplicemente rimossa la colonna D e la riga che mostra D=1. Si può rimuovere anche la colonna CLR e tutte le righe che mostrano CLR=0 poichè è collegato a NOT RESET. NOT RESET sarà sempre ad 1 all'interno di un ciclo di stampa poichè FFA non commuterà ON se NOT RESET è 0.

Per FFB può ora essere impiegata la seguente tabella di funzione semplificata, assumendo che CLR (NOT RESET) sarà sempre 1 e D sarà sempre 0:

FFA ($\overline{Q}$) AND $\overline{\text{RETURN STROBE}}$ =PRESET	FFE ($\overline{Q}$) =CLOCK	Q	$\overline{Q}$
0	0 oppure 1	1	0
1	0 → 1	0	1

Si osservi l'ingresso FFB PRESET; esso è dato da FFA ($\overline{Q}$) AND $\overline{\text{RETURN STROBE}}$.

RIPOSIZIONAMENTO DELLA RUOTA DI STAMPA PER IL CICLO DI STAMPA

Si ricordi che $\overline{\text{RETURN STROBE}}$ è un segnale inviato dalla logica esterna per iniziare uno speciale ciclo di stampa che muove la ruota alla sua posizione di visibilità ma non alimenta il martelletto o stampa un carattere. Chiamiamo questo speciale ciclo di stampa "Riposizionamento della Ruota di stampa". Perciò tra cicli di stampa $\overline{\text{RETURN STROBE}}$ deve essere alto.

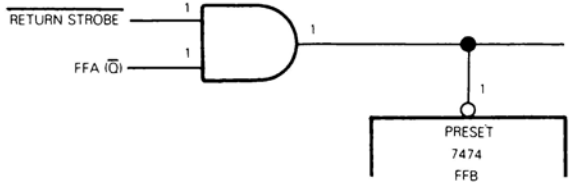
Poichè $\overline{\text{RETURN STROBE}}$ è inviato basso come metodo alternativo per inizializzare un cido di stampa, nella simulazione di FFB si considerano due modi per $\overline{\text{RETURN STROBE}}$

1) Come contributo all'ingresso PRESET.

2) Come segnale che inizia un ciclo di stampa, attraverso un by-pass del flip-flop FFA.

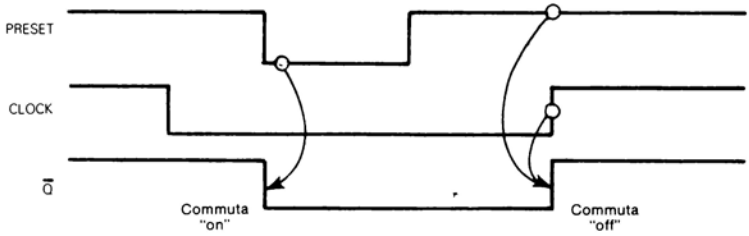
Ma prima si definisce la condizione del flip-flop FFB tra cicli di stampa.

Come è stato appena visto nella simulazione del flip-flop FFA, l'uscita FFA ($\overline{Q}$) è alta fino all'inizio del ciclo di stampa quando $\overline{Q}$ diventa bassa; l'uscita FFA ($\overline{Q}$) è quindi alta tra i cicli di stampa. Per definizione $\overline{\text{RETURN STROBE}}$ è alto tra i cicli di stampa poichè esso è impiegato per iniziare un ciclo di stampa di riposizionamento della ruota. **Perciò l'ingresso FFB PRESET sarà alto tra cicli di stampa:**



Poichè PRESET è alto tra cicli di stampa si assumerà che, iniziando un ciclo di stampa, FFB sia spento; cioè Q sia basso è $\overline{Q}$ alto. Questo assume anche che da qualche tempo PRESET sia alto quando l'uscita $\overline{Q}$ del flip-flop FFE va da 0 ad 1. Come si vedrà in seguito questo è veramente quanto accade in ogni ciclo di stampa.

Perciò iniziando un nuovo ciclo di stampa, FFB ha l'ingresso PRESET alto, con la uscita $\overline{Q}$ alta e Q bassa. Questo flip-flop ora funziona come un interruttore; esso è acceso dal diventare basso di PRESET; viene successivamente acceso dalla transizione da 0 a 1 del clock che si verifica dopo che PRESET è diventato ancora alto:



L'interruttore "acceso" sopra illustrato si verifica in due circostanze:

1) Immediatamente dopo l'inizio di un nuovo ciclo di stampa, quando l'uscita $\overline{Q}$ di FFA è bassa, forzando così basso anche PRESET.

2) Quando $\overline{\text{RETURN STROBE}}$ è inviato basso segnalando un ciclo di riposizionamento della ruota di stampa.

L'interruttore "spento" si verifica quando l'uscita FFE ($\overline{Q}$) subisce una transizione da basso ad alto mentre PRESET sta diventando alto; questo si verifica alla fine di ogni ciclo di stampa.

SIMULAZIONE DEL FLIP-FLOP FFB

COMMUTAZIONE ON DI BIT

Il Bit 1 della Porta I/O 1 è stato assegnato all'uscita Q del flip-flop FFB. L'interruttore "acceso" sopra illustrato è perciò simulato dalle seguenti tre istruzioni:

```
IN    1    ; Carica il byte dei dati del flip-flop
ANI   FDH  ; RESET a 0 del bit 1
OUT   1    ; Ri-immagazzina il byte dati del flip-flop
```

Ecco come lavora l'istruzione ANI:

```
7 6 5 4 3 2 1 0    N. Bit
X X X X X X Y X    Contenuti Accumulatore
1 1 1 1 1 1 0 1    FDH
-----------------
X X X X X X 0 X    AND
```

COMMUTAZIONE OFF DI BIT

Successivamente l'interruttore "spento" può essere così simulato:

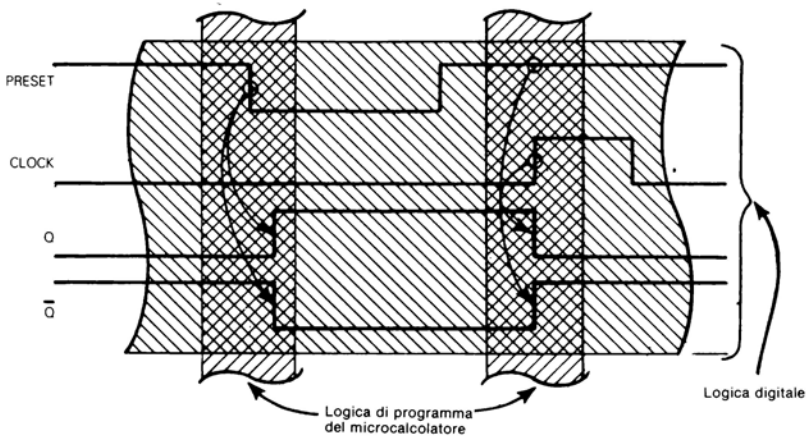
```
IN    1    ; Carica il byte dati del flip-flop
ORI   2    ; Pone ad 1 il bit 1
OUT   1    ; Ri-immagazzina il byte dati del flip-flop
```

Ecco come opera l'istruzione ORI:

```
7 6 5 4 3 2 1 0    N. Bit
X X X X X X Y X    Contenuti Accumulatore
0 0 0 0 0 0 1 0    2
-----------------
X X X X X X 1 X    OR
```

Si incontra ora una situazione nella quale, con ogni miglior intenzione, non si è in grado di simulare direttamente la logica digitale.

E' abbastanza facile disegnare un flip-flop 7474 in un diagramma logico e connettere i suoi pin ai segnali adatti. Avendo fatto questo si richiede di conoscere quando un segnale cambia o no stato. Sfortunatamente una sequenza d'istruzione in linguaggio assembly non ha pin e segnali; **il linguaggio assembly simulerà eventi che si verificano una volta sola ad un certo istante. Per il flip-flop FFB questo può essere illustrato come segue:**



Immediatamente dopo che il flip-flop FFA si accende si entra in un nuovo ciclo di stampa, l'uscita $\overline{Q}$ è bassa, che fa accendere il flip-flop FFB. FFB non si spegnerà che molto più avanti nel ciclo di stampa quando è alta l'uscita $\overline{Q}$ di FFE. **Si deve perciò suddividere la simulazione e di FFB in due parti:**

1) All'Inizio del programma si simulerà FFB commutato on poichè cronologicamente è il successivo evento all'interno del ciclo.
2) Proseguendo nel programma, quando si simula che FFE pone $\overline{Q}$ alto, si deve ricordare di simulare FFB commutato off.

Ma questo non è tutto per la simulazione di FFB. **Si deve anche modificare la sequenza d'istruzione eseguita tra i cicli di stampa, cosicchè $\overline{\text{RETURN STROBE}}$ possa essere simulato iniziando un ciclo di riposizionamento della ruota di stampa.**

Ecco come appare il programma con le istruzioni modificate o nuove riportate in zona oscura:

```
; Esecuzione del programma tra cicli di stampa
; Inizialmente pone a 0 il bit 1, ad 1 il bit 0 della Porta I/O 1
      IN    1     ; Ingresso della Porta I/O 1 all'Accumulatore
      ORI   1     ; Pone il bit 0
      ANI   FDH   ; RESET il bit 1
      OUT   1     ; Ritorno risultato
; Prova per RETURN STROBE basso
L10   IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   10H   ; Isola RETURN STROBE
      JZ    FFB   ; Se è 0 salta alla simulazione di FFB
; Simulazione di FFA e logica associata
; Carica i contenuti della Porta I/O 2 nell'Accumulatore ed
; Isola i bit 1, 5 e 6 per CH RDY, PW STROBE
; e RESET, rispettivamente
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   62H   ; Isola i bit 6, 5 ed 1
      CPI   22H   ; Se RESET=0, CH RDY=1 e PW STROBE=1
      JNZ   L10   ; Inizia un nuovo ciclo di stampa altrimenti ritorna ad L10
      IN    1     ; Per partire un nuovo ciclo di stampa pone
      ANI   FEH   ; a 0 il bit 0 della Porta I/O 1
      OUT   1
; La sequenza del nuovo ciclo di stampa inizia qui
; Simula la commutazione on del flip-flop
FFB   IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   1     ; Ri-immagazzina il risultato
```

La simulazione del flip-flop FFB non termina qui. Si osservi che l'uscita $\overline{Q}$ del flip-flop FFB è collegata.

1) Ad un gate AND 7411, localizzato approssimativamente alle coordinate B5.

2) Ad un gate OR 7432, localizzato a C7.

L'uscita FFB (Q) non è inutile ma servirà in seguito.

Prima si considera il gate AND 7411 localizzato a B5.

Se si ritorna alla descrizione dei segnali d'uscita si noterà che CH RDY è stato dichiarato essere alto tra cicli di stampa ma basso durante un ciclo di stampa.

In realtà CH RDY esce dal gate 7411 AND localizzato a B5, perciò, tra i cicli di stampa, tutti questi tre ingressi al suddetto gate AND devono essere alti. L'analisi del flip-flop FFB mostra la sua uscita $\overline{Q}$ infatti sarà alta tra cicli di stampa, ma per il momento si deve prendere questo come un dato di fatto che gli altri due segnali d'ingresso del gate AND saranno alti anche tra i cicli di stampa.

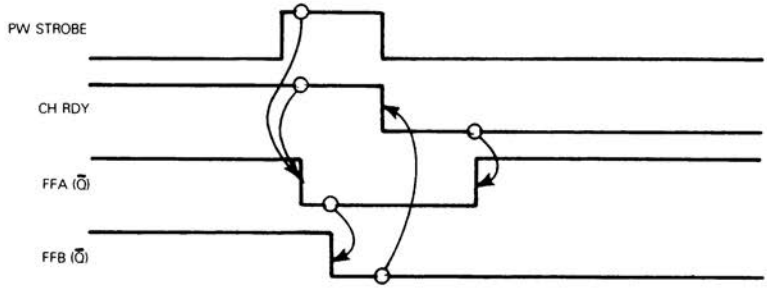
In ogni caso, non appena il flip-flop FFB commuta on, la sua uscita $\overline{Q}$ diventa bassa che significa non esistono dati dagli altri due ingressi del gate AND 7411, così anche CH RDY sarà guidato basso. Questo cambiamento nello stato di CH RDY è simulato dall'aggiunta delle seguenti due istruzioni al programma precedente:

```
; Prova per RETURN STROBE basso
L10   IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   10H   ; Isola RETURN STROBE
      JZ    FFB   ; Se questo è 0 salta alla simulazione FFB
; Simulazione di FFA e della logica associata
; Carica i contenuti della Porta I/O 2 nell'Accumulatore ed
; Isola i bit 1, 5 e 6 per CH RDY, PW STROBE
; e RESET rispettivamente
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   62H   ; Isola i bit 6, 5 ed 1
      CPI   22H   ; Se RESET=0, CH RDY=1 e PW STROBE=1,
      JNZ   L10   ; Inizia un nuovo ciclo di stampa, altrimenti ritorna ad L10
      IN    1     ; Per iniziare il posizionamento di un nuovo ciclo stampa
      ANI   FEH   ; Pone a 0 il bit 0 della Porta I/O 1
; La nuova sequenza del ciclo di stampa comincia qui
; Simula il flip-flop FFB che commuta on
FFB   IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   1     ; Ri-immagazzina il risultato
; Simula il Gate 7411 che commuta basso CH RDY
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   2     ; Ri-immagazzina il risultato
```

Si è ora di fronte ad un problema interessante. CH RDY diviene l'ingresso D del flip-flop FFA ed esso contribuisce ad un ingresso CLR di FFA. **Cosa succede quando CH RDY diventa basso in risposta alla commutazione ON di FFB?**

Si noti che PW STROBE pulsa soltanto alto perciò il gate OR localizzato alle coordinate B2 fa affidamento su CH RDY che è alto in modo da fornire un ingresso alto al gate AND successivo. Questo gate AND, a sua volta, fornisce un'ingresso CLR alto al flip-flop FFA. In altre parole all'istante in cui il flip-flop FFB commuta ON e commuta basso CH RDY, PW STROBE sarà già basso; così entrano PW STROBE e CH RDY entrambi bassi. **Se si rivede la tabella della verità di CLR del flip-flop FFA si troverà che quando CH RDY e PW STROBE sono entrambi 0, CLR sarà sempre 0.**

Perciò il flip-flop FFA commuterà off:



Cosa significa questo? Si conclude che il flip-flop FFA commuta "on" sé stesso allo inizio di un ciclo di stampa e rimane on quanto basta per commutare on FFB. Quando FFB commuta con esso pone basso CH RDY e questo commuta off il flip-flop FFA.

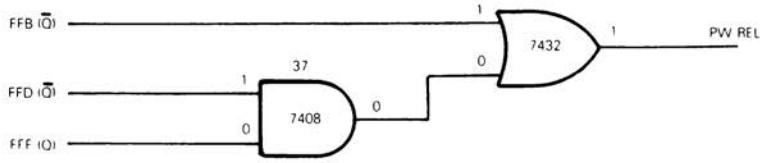
TIMING E SEQUENZA LOGICA

Ma ecco una difficoltà: se si guarda ancora la Figura 3-1 si troverà che il flip-flop FFA aiuta a generare l'ingresso J del flip-flop FFC oltre alla commutazione on del flip-flop FFB.

Ora poichè gli eventi sono seriali nel tempo si può andare avanti e simulare la commutazione off del flip-flop FFA, così come si ricorderà dalla simulazione del flip-flop FFC, dato che esso riceve $\overline{Q}$ bassa dal flip-flop FFA.
Ricordando questa precauzione si estenderà il programma come segue:

```
; Prova per RETURN STROBE basso
L10   IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   10H   ; Isola RETURN STROBE
      JZ    FFB   ; Se esso è 0, salta alla simulazione di FFB
; Simulazione di FFA e della logica associata
; Carica i contenuti della Porta I/O 2 nell'Accumulatore ed
. Isola i bit 1, 5 e 6 CH RDY, PW STROBE
; e RESET rispettivamente
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   62H   ; Isola i bit 6, 5 ed 1
      CPI   22H   ; Se RESET=0, CH RDY=1 e PW STROBE=1
      JNZ   L10   ; Inizia un nuovo ciclo di stampa altrimenti ritorna ad L10
      IN    1     ; Per iniziare un nuovo ciclo di stampa, pone
      ANI   FEH   ; a 0 il bit 0 della Porta I/O 1
      OUT   1
; La sequenza del nuovo ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta on
FFB   IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   1     ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
      IN    2     ; Carica la Porta I/O 2 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   2     ; Ri-immagazzina il risultato
CH RDY basso commuta OFF FFA. Pone ad 1 il bit 0 della Porta I/O 1
      IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ORI   1     ; Pone il bit 0 ad 1
      OUT   1     ; Ri-immagazzina il risultato
```

Ora si osservi il gate OR localizzato alle coordinate C7. Questo gate riceve FFB $\overline{Q}$ come uno dei suoi ingressi per generare PW REL. L'altro ingresso di questo gate OR è l'AND dell'uscita Q dal flip-flop FFF, più l'uscita $\overline{Q}$ del flip-flop FFD. Si troverà brevemente che questi flip-flop sono anche commutati off tra cicli di stampa; essi sono commutati on sequenzialmente nel corso del ciclo di stampa. All'istante in cui FFB commuta on, FFF sarà commutato off che significa che la sua uscita Q sarà bassa, così il gate AND localizzato in C7 avrà uscita bassa. Questo significa che il **gate OR 26 è stato collegato all'uscita $\overline{Q}$ da FFB in modo che PW REL risulta alto:**



Ora quando FFB commuta on e l'uscita $\overline{Q}$ è bassa, PW REL sarà anch'esso basso. Si deve perciò modificare il programma perchè i bit 0 ed 1 della Porta I/O 3 escano bassi, poichè PW REL e CH RDY devono essere guidati bassi entrambi. Ecco come appare ora il programma:

```
; Prova per RETURN STROBE basso
L10   IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   10H   ; Isola RETURN STROBE
      JZ    FFB   ; Se esso è 0, salta alla simulazione di FFB
; Simulazione di FFA e della logica associata
; Carica i contenuti della Porta I/O 2 nell'Accumulatore ed
; Isola i bit 1, 5 e 6 per CH RDY, PW STROBE
; E RESET, rispettivamente
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   62H   ; Isola i bit 6, 5 ed 1
      CPI   22H   ; Se RESET=0, CH RDY=1 e PW STROBE=1
      JNZ   L10   ; Inizia un nuovo ciclo di stampa, altrimenti ritorna ad L10
      IN    1     ; Per iniziare un nuovo ciclo di stampa, pone
      ANI   FEH   ; a 0 il bit 0 della Porta I/O 1
      OUT   1
; La sequenza del nuovo ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta ON
FFB   IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   1     ; Ri-immagazzina il risultato
; Simula il gate 7411 che commuta basso CH RDY
; Simula inoltre il gate OR 7432 che commuta basso PW REL
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   FCH   ; RESET a 0 i bit 0 ed 1
      OUT   2     ; Ri-immagazzina il risultato
; CH RDY basso commuta OFF FFA, pone ad 1 il bit 0 della Porta I/O 1
      IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ORI   1     ; Pone ad 1 il bit 0
      OUT   1     ; Ri-immagazzina il risultato
```

Non c'è nulla da fare sull'uscita Q proveniente dal flip-flop FFB? Se si osserva questa uscita si vedrà che essa collega direttamente gli ingressi RESET dei flip-flop FFC, FFD ed FFE. Essa diventa anche uno degli ingressi del multivibratore 555.

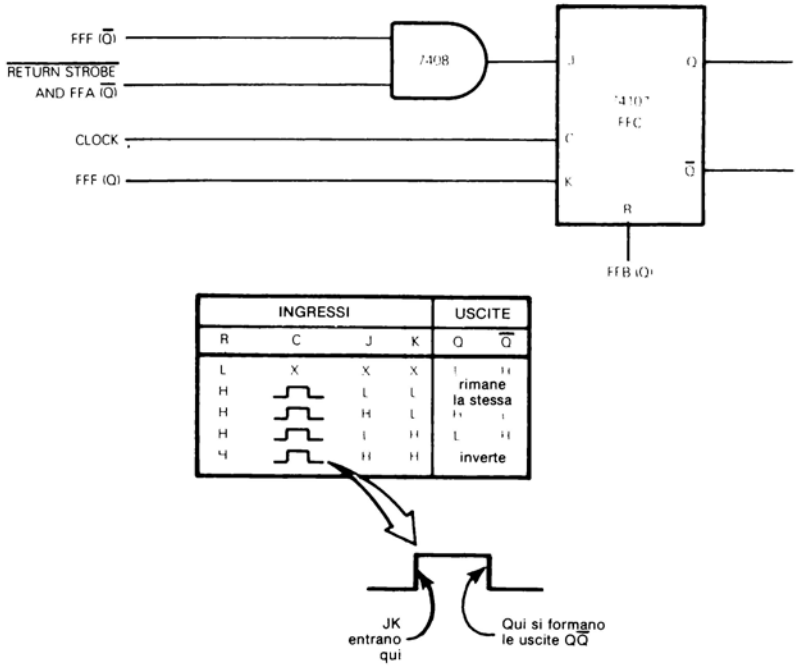
Infatti l'uscita FFB Q è un segnale limitante se basso, esso chiude off i quattro dispositivi connessi; quando è alto questi quattro dispositivi sono commutati on.

L'uscita FFB Q sarà presa in considerazione quando si simulano i quattro dispositivi connessi a questo segnale. Perciò la simulazione del flip-flop FFB è così completa.

FLIP-FLOP FFC

Questo è il flip-flop 74107 di coordinate C2 in Figura 3-1. Poichè si stanno simulando quattro flip-flop 74107 ci si può riferire al Capitolo 2 per ricordare immediatamente le caratteristiche di questo dispositivo.

Si isoli il flip-flop FFC per vedere come lavora:

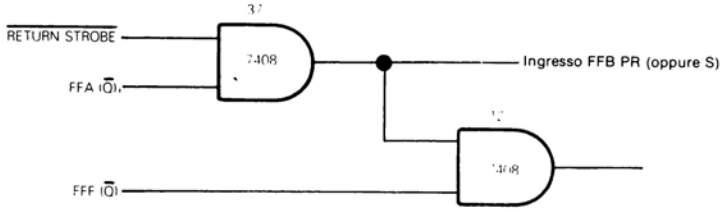


Tra i cicli di stampa, l'uscita Q di FFB, essendo bassa, commuta off il flip-flop FFC. Perciò FFC ha le uscite Q bassa e $\overline{Q}$ alta.

Vediamo cosa accade quando FFB è commutato on in relazione agli ingressi J e K che arrivano ad FFC.

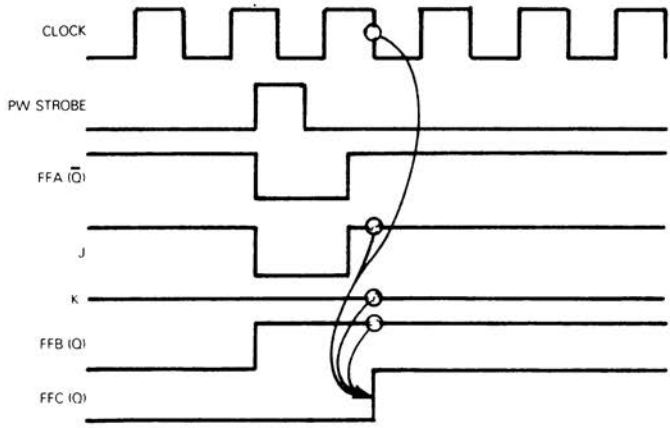
Tra i cicli di stampa il flip-flop FFF è commutato on, perciò la sua uscita Q sarà bassa. FFC riceve il suo ingresso K dall'uscita FFF Q, perciò quando FFC commuta on, il suo ingresso K sarà 0.

L'ingresso J di FFC è generato come segue:



FFF ($\overline{Q}$) sarà alta poichè FFF è commutato off. l'ingresso J di FFC sarà identico all'ingresso FFB PR che è stato già descritto.

Riassumendo, questa è la sequenza di segnale che commuta on FFC:



Quando l'uscita FFB Q diventa alta, non limitando FFC, FFC attende finchè l'uscita FFA $\overline{Q}$ diventa alta nuovamente; allora FFC riceverà un ingresso alto in J ed uno basso in K. Quando il gradino di coda del successivo impulso di clock entra in FFC, Q uscirà alta e $\overline{Q}$ bassa.

FFC attende che FFA $\overline{Q}$ esca ancora alta, perchè mentre FFA è commutato on, $\overline{Q}$ esce bassa. Mentre FFA ($\overline{Q}$) (oppure $\overline{\text{RETURN STROBE}}$) è pulsato basso, FFC riceve un ingresso J basso. Così finchè FFC sta ricevendo gli ingressi J e K bassi, la sua uscita non cambia — questa è una delle proprietà del flip-flp 74107.

Il flip-flop FFC rimarrà nel suo stato on finchè il flip-flop FFF commuta on, in un punto successivo del ciclo di stampa. In quell'istante il flip-flop FFC riceverà un ingresso alto in K ed uno basso in J; questo originerà la commutazione off di FFC.

SIMULAZIONE DEL FLIP-FLOP FFC

La simulazione del flip-flop è diretta; essa comprende queste tre fasi:

1) **Si devono regolare le istruzioni di inizializzazione per assicurare che il flip-flop FFC sia riportato off tra i cicli di stampa.**
2) **La simulazione del flip-flop FFB deve essere seguita immediatamente da istruzioni che simulano la commutazione on del flip-flop FFC.**
3) **Si deve ricordare di simulare la commutazione off di FFC — ma questo non accadrà che ad una fase successiva del programma.**

Ora le seguenti modifiche all'inizio del programma assicurano che il flip-flop FFC è simulato off tra cicli di stampa:

```
; Esecuzione del programma tra cicli di stampa
; Inizialmente pone i bit 2 ed 1 della Porta I/O a 0, il bit 0 ad 1
      IN    1     ; Ingresso della Porta I/O 1 all'Accumulatore
      ORI   1     ; Pone il bit 0
      ANI   F9H   ; RESET i bit 2 ed 1
      OUT   1     ; Ritorno del risultato
; Prova per RETUNR STROBE basso
L10   IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   10H   ; Isola RETURN STROBE
      JZ    FFB   ; Se esso è 0 salta alla simulazione di FFB
```

Sostanzialmente è stata modificata la maschera dell'AND per comprendere il bit 2 della Porta I/O tra i bit ripristinati a 0:

		Contenuto Accumulatore
		7 6 5 4 3 2 1 0 ← N. Bit
IN	1	X X X X X X X X
ORI	1	0 0 0 0 0 0 0 1
		X X X X X X X 1
ANI	F9H	1 1 1 1 1 0 0 1
		X X X X X 0 0 1

Si ricordi che il bit 2 della Porta I/O è stato assegnato al flip-flop FFC.

TIMING E SEQUENZA LOGICA

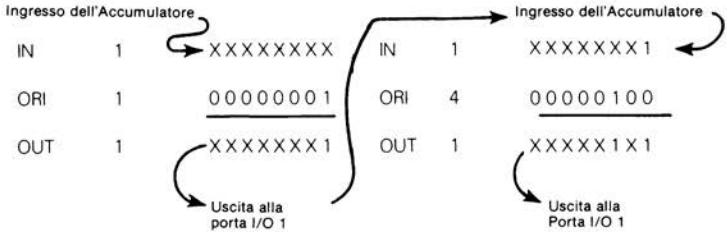
E per quanto riguarda il ritardo di tempo che separa la commutazione on dei flip-flop B e C? Si ricordi che il flip-flop FFC non commuterà on se non dopo che il flip-flop FFB ha commutato off il flip-flop FFA. Se si tratta di un riposizionamento della ruota di stampa FFC non commuterà on finchè RETURN STROBE non è alto.

La semplicità o complessità dei problemi di timing dipende interamente dalla logica al di là della Figura 3-1. Non c'è niente all'interno della logica della Figura 3-1 che imponga un ritardo di tempo di durata prefissata oppure, per quanto ci riguarda, qualsiasi ritardo di tempo separante la commutazione on dei flip-flop FFB ed FFC. Quindi non si porrà attenzione alle considerazioni di timing associate con la commutazione on di FFC, invece si **aggiungeranno semplicemente delle istruzioni di simulazione alla fine del programma nel modo seguente:**

```
; La nuova sequenza del ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta on
FFB  IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
     ANI   FDH   ; RESET a 0 il bit 1
     OUT   1     ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Inoltre simula il gate OR 7432 che commuta basso PW REL
     IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
     ANI   FCH   ; RESET a 0 i bit 0 ed 1
     OUT   2     ; Ri-immagazzina il risultato
; CH RDY basso commuta off FFA, pone ad 1 il bit 0 della Porta I/O 1
     IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
     ORI   1     ; Pone il bit 0 ad 1
     OUT   1     ; Ri-immagazzina il risultato
; Simula il flip-flop FFC 74107 che commuta on
; Pone ad 1 il bit 2 della Porta I/O 1
     IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
     ORI   4     ; Pone il bit 2 ad 1
     OUT   1     ; Ri-immagazzina il risultato
```

PROGRAMMI RESI PIU' BREVI

Se si è pervenuti a pensare come un programmatore si riconoscerà l'opportunità di economizzare nella simulazione della commutazione on del flip-flop FFC. **Si osservi che le precedenti tre istruzioni indicate con Ⓑ costituiscono anche il posizionamento ad 1 di un bit dell porta I/O 1.** Questo genera la seguente sequenza di eventi:



Si può combinare le due operazioni come segue:

```
IN    1    XXXXXXXX
ORI   5    00000101
           XXXXX1X1
```

Le istruzioni segnate Ⓑ vengono ora sostituite da quelle indicate con Ⓒ :

```
; Simula il flip-flop FFB che commuta on
FFB  IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
     ANI   FDH   ; RESET a 0 il bit 1
     OUT   1     ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Inoltre simula il gate OR 7432 che commuta basso PW REL
     IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
     ANI   FCH   ; RESET a 0 i bit 0 ed 1
     OUT   2     ; Ri-immagazzina il risultato
. CH RDY basso commuta off FFA. Pone ad 1 il bit 0 della Porta I/O 1
; Inoltre simula FFC che commuta on. Pone ad 1 il bit 2 della Porta I/O 1
     IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
Ⓒ    ORI   5     ; Pone ad 1 i bit 2 e 0
     OUT   1     ; Ri-immagazzina il risultato
```

La simulazione del flip-flop C che commuta on è così completa.

SIMULAZIONE DELL'IMPULSO DI INIZIO MOVIMENTO DEL NASTRO

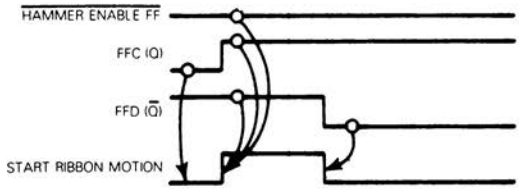
Si ricordi che inizialmente in un ciclo di stampa il segnale d'uscita dell'INIZIO MOVIMENTO DEL NASTRO è mantenuto alto per far scattare la logica esterna che avanza il nastro, così quando il martelletto viene alimentato di fronte al carattere da stampare viene a trovarsi del nastro fresco. Il segnale di INIZIO MOVIMENTO DEL NASTRO è generato da un gate AND 7411 (numero 7) localizzato alle coordinate C7 in Figura 3-1. Questo gate AND ha tre ingressi:

1) $\overline{\text{HAMMER ENABLE FF}}$. Questo è un segnale d'ingresso per identificare un ciclo di riposizionamento della ruota di stampa.
2) L'uscita Q dal flip-flop FFC.
3) L'uscita $\overline{Q}$ dal flip-flop FFD.

$\overline{\text{HAMMER ENABLE FF}}$ sarà alto a meno che non sia in corso un ciclo di riposizionamento della ruota, nel quale caso il nastro non deve essere mosso. Perciò questo segnale sopprime l'impulso di INIZIO MOVIMENTO DEL NASTRO.

Tra cicli di stampa i flip-flop FFC ed FFD sono entrambi commutati off; perciò FFC (Q) è bassa ed FFD ($\overline{Q}$) è alta.

L'uscita FFC (Q) mantiene basso il segnale d'INIZIO MOVIMENTO DEL NASTRO. Quando FFC commuta on durante un normale ciclo di stampa, tutti gli ingressi al gate AND 7 saranno alti cosicchè sarà alto anche l'impulso d'INIZIO MOVIMENTO DEL NASTRO; esso rimarrà alto finchè il flip-flop FFD non commuta on ed in quell'istante FFD avrà uscita $\overline{Q}$ bassa; e questo farà andare al livello basso l'impulso d'INIZIO MOVIMENTO DEL NASTRO. **Il timing può essere illustrato come segue:**



Se si osserva il diagramma di timing illustrato in Figura 3-2 si vedrà che l'impulso di uscita di INIZIO MOVIMENTO DEL NASTRO è estremamente breve. Perciò invece di impiegare il flip-flop FFD per regolare la fine dell'IMPULSO ALTO DI INIZIO MOVIMENTO DEL NASTRO, **si eseguiranno semplicemente le istruzioni per commutare on il bit 3 della Porta I/O C e poi per commutarlo immediatamente off, come segue:**

```
; La sequenza del nuovo ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta on
FFB   IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ANI   FDH   ; RESET a 0 il bit 1
      OUT   1     ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Simula anche il gate OR 7432 che commuta basso PW REL
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   FCH   ; RESET a 0 i bit 0 ed 1
      OUT   2     ; Ri-immagazzina il risultato
; CH RDY basso commuta off FFA. Pone ad 1 il bit 0 della Porta I/O 1
; Inoltre simula FFC che commuta on. Pone ad 1 il bit 2 della Porta I/O 1
      IN    1     ; Carica la Porta I/O 1 nell'Accumulatore
      ORI   5     ; Pone ad 1 i bit 2 e 0
      OUT   1     ; Ri-immagazzina il risultato
; Impulso alto d'INIZIO MOVIMENTO DEL NASTRO
      IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      ORI   8     ; Pone alto il bit 3
      OUT   2     ; Uscita del risultato
      ANI   F7H   ; Commuta off il bit 3
      OUT   2     ; Uscita del risultato
```

CALCOLO DELLA LARGHEZZA D'IMPULSO

Si può calcolare la larghezza d'impulso d'INIZIO MOVIMENTO DEL NASTRO sommando i tempi di esecuzione dell'istruzione tra il posizionamento alto e basso del pin 3 della Porta I/O 2:

```
Cicli   Istruzione
 10     OUT   2     ;USCITA RISULTATO
  7     ANI   F7H   ;COMMUTA OFF IL BIT 3
 10     OUT   2     ;USCITA RISULTATO
```

Larghezza d'impulso = 17 cicli, oppure 8.5 microsecondi impiegando un clock di 500 nanosecondi

Cosa succede in seguito? La sequenza logica deve considerare il flip-flop FFD, alla destra di FFC oppure si può finire sul 74121 one-shot numero 36, proprio in basso a destra di FFC.

L'One-shot 36 ha due ingressi A collegati a massa, cioè essi saranno al livello basso. Se si osserva la tabella di funzione fornita al Capitolo 2, si troverà che in questa configurazione un'uscita one-shot è fatta scattare da una transizione di B da basso-ad-alto. FFC ($\overline{Q}$) fornisce il trigger. Qualsiasi altro ingresso B manterrà spenta quest'uscita one-shot — questo significa che **Q e $\overline{Q}$ saranno bassa ed alta rispettivamente, fino a molto più avanti nel ciclo di stampa, quando FFC commuta off;** questo si verifica quando l'uscita FFC $\overline{Q}$ è soggetta ad una transizione da basso ad alto.

Il flip-flop FFD diviene il dispositivo successivo da simulare.

FLIP-FLOP FFD

Il flip-flop FFD riceve il suo ingresso J direttamente dall'uscita FFC (Q); inoltre riceve il suo ingresso K dall'uscita FFC ($\overline{Q}$). Si ricordi che, poichè one-shot 36 è ancora commutato off, la sua uscita $\overline{Q}$ sarà alta; questo significa che il gate AND 12 consentirà semplicemente ad FFC ($\overline{Q}$) di propagarsi direttamente per arrivare all'ingresso FFD (K).

Ora il flip-flop FFD riceve gli stessi segnali di clock e reset di FFC, perciò il **flip-flop FFD commuterà on un ciclo di clock più tardi rispetto al flip-flop FFC.**

SIMULAZIONE DEL FLIP-FLOP FFD

La simulazione del flip-flop FFD è pressochè identica alla simulazione del flip-flop FFC, la differenza principale è che il bit 3 della Porta I/O 1 è stato assegnato al flip-flop FFD. Ancora una volta occorre limitare la commutazione on del flip-flop FFD ed assicurarsi che il suo posizionamento tra cicli di stampa sia corretto.

Il flip-flop FFD è commutato off più tardi nel ciclo di stampa; si deve perciò tener conto di questo fatto nel programma.

Ecco le aggiunte e le modifiche necessarie al programma:

```
; Esecuzione del programma tra cicli di stampa
  Inizialmente pone a 0 i bit 3, 2 ed 1 ed a 1 il bit 0 della Porta I/O
      IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
(D)   ORI     1       ; Pone il bit 0
      ANI     F1H     ; RESET i bit 3, 2 ed 1
      OUT     1       ; Ritorno risultato
; Prova per RETURN STROBE basso
L10   IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI     10H     ; Isola RETURN STROBE
      JZ      FFB     ; Se esso è 0 salta alla simulazione di FFB
      —
      —
      —
; Simula il flip-flop FFB che commuta on
FFB   IN      1       ; Carica la Porta I/O 1 nell'Accumulatore
      ANI     FDH     ; RESET a 0 il bit 1
      OUT     1       ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Inoltre simula il gate OR 7432 che commuta basso PW REL
      IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI     FCH     ; RESET a 0 i bit 0 ed 1
      OUT     2       ; Ri-immagazzina il risultato
; CH RDY basso commuta off FFA. Pone ad 1 il bit 0 della Porta I/O 1
```

```
; Inoltre simula FFC che commuta on. Pone ad 1 il bit 2 della Porta I/O 1
        IN      1       ; Carica la Porta I/O 1 nell'Accumulatore
        ORI     5       ; Pone ad 1 i bit 2 e 0
        OUT     1       ; Ri-immagazzina il risultato
; Impulso d'INIZIO MOVIMENTO DEL NASTRO al livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     8       ; Pone alto il bit 3
        OUT     2       ; Uscita risultato
        ANI     F7H     ; Commuta basso il bit 3
        OUT     2       ; Uscita risultato
; Simula FFD che commuta on. Pone ad 1 il bit 3 della Porta I/O 1
        IN      1       ; Carica la Porta I/O 1 nell'Accumulatore
        ORI     8       ; Pone ad 1 il bit 3          <---- (E)
        OUT     1       ; Ri-immagazzina il risultato
```

Se le aggiunte e le modifiche al programma sopra illustrate non sono immediatamente ovvie, si confrontino le stesse con quelle della simulazione del flip-flop C. Non si prosegua se non si sono comprese le variazioni al programma del flip-flop FFD.

PROGRAMMI RESI PIU' BREVI

Come la simulazione dell'FFC che commuta on (B) è assorbita nella simulazione di FFB ((C)), **così la simulazione di FFD che commuta on ((E)) può essere assorbita come segue:**

```
; La nuova sequenza del ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta on
FFB     IN      1       ; Carica la Porta I/O 1 nell'Accumulatore
        ANI     FDH     ; RESET a 0 il bit 1
        OUT     1       ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Inoltre simula il gate OR 7432 che commuta basso PW REL
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ANI     FCH     ; RESET a 0 i bit 0 ed 1
        OUT     2       ; Ri-immagazzina il risultato
; CH RDY commuta off FFA. Pone ad 1 il bit 0 della Porta I/O 1
; Inoltre simula FFC ed FFD che commutano on. Pone ad 1 i bit 3 e 2 della Porta I/O 1
        IN      1       ; Carica la Porta I/O 1 nell'Accumulatore
(F)-->  ORI     0DH     ; Pone ad 1 i bit 3, 2 e 0
        OUT     1       ; Ri-immagazzina il risultato
; Impulso d'INIZIO MOVIMENTO DEL NASTRO al livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     8       ; Pone alto il bit 3
        OUT     2       ; Uscita risultato
        ANI     F7H     ; Commuta off il bit 3
        OUT     2       ; Uscita risultato
```

Se la simulazione è combinata ((F)), i flip-flop FFC ed FFD commuteranno on esattamente nello stesso istante.

La logica della Figura 3-1 mostra FFD che commuta on un impulso di clock dopo FFC. Se il periodo di clock è 2 microsecondi allora ci sarà un ritardo di due microsecondi tra la commutazione on dei flip-flop FFD ed FFC. Entrambe le simulazioni sono errate.

TIMING E LIMITI DI SIMULAZIONE

Perchè? Onestamente non si può dire come viene manipolata l'informazione. Infatti non si conosce come la logica esterna impiega le uscite FFC ed FFD. **Se l'intervallo di tempo di commutazione tra questi due flip-flop deve essere minore di 2 microsecondi, allora questa simulazione non è operativa.** Entrambi i flip-flop devono divenire parte della "logica esterna" oppure si deve trovare qualche altro mezzo di simulazione dell'eventuale funzione globale.

Se la logica esterna richiede qualche ritardo di commutazione, ma non ci sono specifiche stringenti sulla lunghezza del ritardo di tempo, allora la simulazione del flip-flop FFD è adeguata.

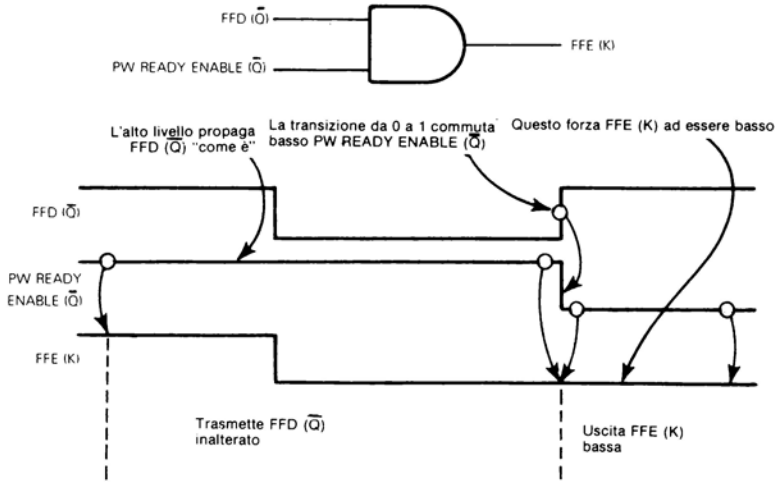
E' affatto possibile che la logica della Figura 3-1 mostri un ritardo di tempo di commutazione tra i flip-flop FFC ed FFD solo per definire i gradini di testa e di coda dell'impulso d'INIZIO MOVIMENTO DEL NASTRO; ma ci si è assicurati che questo impulso sia alto dalle istruzioni di esecuzione sequenziale che fanno uscire 1 e poi 0 dal bit 3 della Porta I/O 2. Così lontano per quanto riguarda la logica interna della Figura 3-1 quindi scompare la necessità di un ritardo di tempo tra i flip-flop FFC ed FFD. **In questo caso si assumerà che la logica esterna non richieda un ritardo di tempo di commutazione tra i flip-flop FFC ed FFD; inoltre si adotterà la simulazione combinata più breve identificata mediante Ⓕ .**

FLIP-FLOP FFE

Il dispositivo successivo nella sequenza logica è il flip-flop FFE. La circuitazione attorno a questo flip-flop è pressochè identica ad FFD.

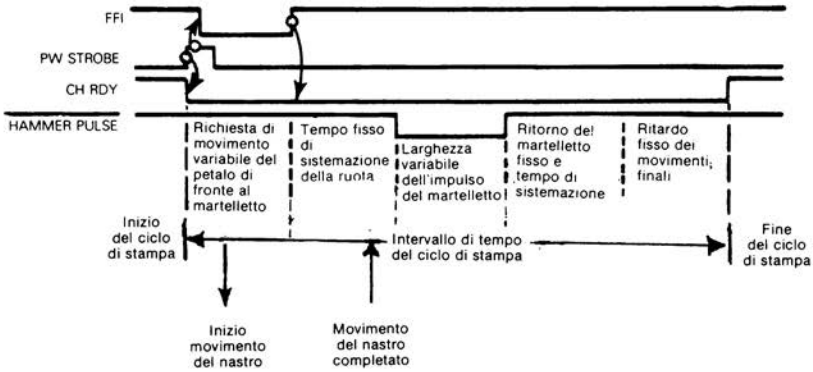
L'ingresso FFE (K) è collegato all'uscita FFD ($\overline{Q}$) interrotta da un altro componente del gate AND 12. L'altro ingresso di questo AND è l'uscita $\overline{Q}$ dell'one-shot 49. L'one-shot 49 è collegato nello stesso modo dell'one-shote 36 appena descritto.

La transizione dell'uscita $\overline{Q}$ del flip-flop FFD da 0 ad 1 ricorrerà quando FFD è commutato off e questa è la transizione che farà scattare lo one-shote 49. **Perciò lo one-shote 49 avrà uscita $\overline{Q}$ alta finchè il flip-flop FFD è commutato off, che significa che quando FFD commuta on, la sua uscita $\overline{Q}$ si propagherà direttamente attraverso il gate AND connettendolo all'ingresso FFE (K):**



L'unica caratteristica del flip-flop FFE è il modo in cui è generato il suo ingresso J. Questo ingresso è l'AND dell'uscita FFD (Q) e del segnale d'ingresso FFI. Ora l'uscita Q di FFD sarà alta non appena FFD commuta on, ma **FFI entra bassa dall'inizio del ciclo di stampa finchè la ruota di stampa non è posizionata correttamente.** (La fun-

zione di questo segnale d'ingresso è stata descritta precedentemente nel capitolo). **Il timing associato con FFI può essere illustrato come segue:**



Mentre FFI è basso, il flip-flop FFE riceverà un ingresso J basso; gli ingressi J e K bassi, come si ricorderà, mantengono le uscite Q di un flip-flop 74107 nelle loro precedenti condizioni. **Così il segnale d'ingresso FFI è stato impiegato per creare il primo ritardo di tempo del ciclo di stampa: un ritardo di tempo variabile è necessario per muovere il petalo della ruota richiesto di fronte al martelletto di stampa. La simulazione di questo ritardo di tempo è abbastanza semplice; essa può essere illustrata come segue:**

```
Impulso d'INIZIO MOVIMENTO DEL NASTRO a livello alto
      IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ORI     8       ; Pone il bit 3 alto
      OUT     2       ; Uscita risultato
      ANI     F7H     ; Commuta off il bit 3
      OUT     2       ; Uscita risultato
; Prova ingresso decodifica di velocità per originare il ritardo del movimento della
; ruota di stampa
VLDC IN       0       ; Ingresso della Porta I/O 0 all'Accumulatore
      RLC             ; Sposta il bit 7 in Carry
      JNC     VLDC    ; Permane nell'anello se Carry è 0
; Alla fine del ritardo simula FFE che commuta on
      IN      1       ; Ingresso della Porta I/O 1
      ANI     DFH     ; RESET il bit 5
      ORI     10H     ; Pone bit 4
      OUT     1       ; Uscita del risultato
```

RITARDO DI TEMPO DI LUNGHEZZA VARIABILE

Allo scopo di generare il ritardo di tempo iniziale, si esegue semplicemente un programma ad anello continuo che fa entrare i contenuti della Porta I/O 0 nell'Accumulatore. Il bit 7 della Porta I/O 0 è stato assegnato al segnale d'ingresso FFI. Si verifica questo bit spostandolo nello stato Carry. Quindi, se lo stato Carry ha un contenuto uguale a 0, FFI deve ancora essere

SALTA SU NO CARRY

basso; così rimane all'interno dell'anello. Non appena un 1 viene spostato nello stato Carry, l'istruzione JNC creerà un risultato "falso"; viene eseguita l'istruzione sequenzialmente successiva e

si esce dall'anello del ritardo di tempo:

```
VLDC        IN      0
            RLC
            JNC     VLDC      C = 0
C = 1       IN      10H
            ANI     DFH
```

Salta su Not Carry significa salta se Carry è 0 (NOT 1). "Salta" significa "non andare alla istruzione sequenzialmente successiva" invece va a VLDC

Le ultime quattro istruzioni della simulazione di FFE mostrano che entrambe le uscite di questo flip-flop diventano segnali d'uscita. Questo è in accordo con la Figura 3-1. Quindi si ripristina il bit 5 (esso rappresenta l'uscita $\overline{Q}$) e si pone ad 1 il bit 4 (esso rappresenta l'uscita Q).

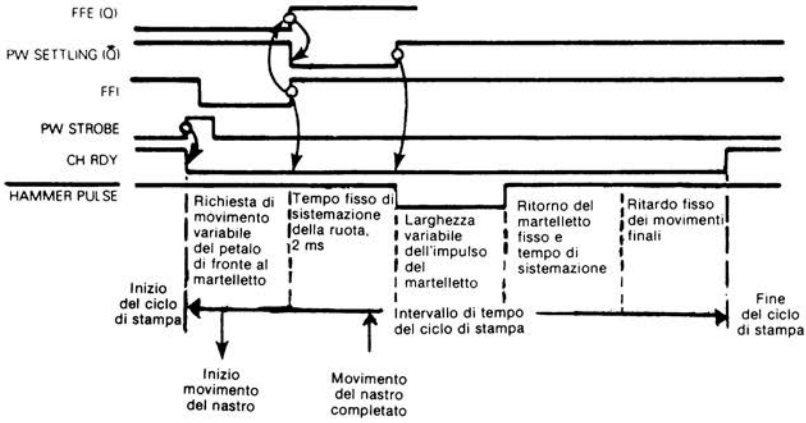
La sequenza d'istruzione eseguita tra cicli di stampa dovrà essere modificata per assicurare che il bit 5 è stato posto inizialmente ad 1, mentre il bit 4 è stato inizialmente ripristinato a 0. Ecco le modifiche necessarie:

```
; Esecuzione del programma tra cicli di stampa
; Inizialmente pone a 0 i bit, 4, 3, 2 ed 1, ad 1 il bit 0 della Porta I/O
      IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
      ORI     21H     ; Pone ad 1 i bit 5 e 0
      ANI     E1H     ; RESET a 0 i bit 4, 3, 2 ed 1
      OUT     1       ; Ritorno risultato
; Prova per RETURN STROBE basso
L10   IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI     10H     ; Isola RETURN STROBE
      JZ      FFB     ; Se esso è 0 salta alla simulazione di FFB
```

ONE-SHOT PW SETTLING

Lo one-shot PW SETTLING è il dispositivo 74121 di coordinate B5 in Figura 3-1. Questo dispositivo è stato descritto al Capitolo 2. Con i suoi due ingressi A collegati a massa **questo one-shot è fatto scattare da una transizione da basso ad alto al suo ingresso B. Poichè l'ingresso B è collegato all'uscita FFE Q, questa transizione si verificherà non appena il flip-flop FFE commuta on.**

Lo one-shot PW SETTLING ha un ritardo di due millisecondi. Questo ritardo deriva dalla combinazione esterna condensatore/resistenza indicati C1 ed R1. Perciò non appena FFE commuta on, lo one-shot PW SETTLING fa uscire $\overline{Q}$ bassa per due millisecondi.



SIMULAZIONE DELLO ONE-SHOT PW SETTLING

SIMULAZIONE DEL RITARDO DI TEMPO ONE-SHOT

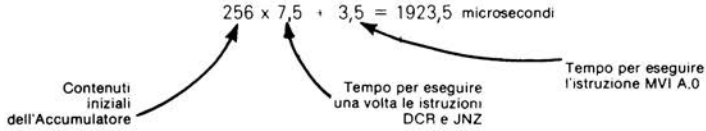
La simulazione del ritardo di tempo one-shot è abbastanza semplice e può essere illustrata come segue:

```
; Impulso d'INIZIO MOVIMENTO DEL NASTRO a livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     8       ; Pone alto il bit 3
        OUT     2       ; Uscita risultato
        ANI     F7H     ; Commuta off il bit 3
        OUT     2       ; Uscita risultato
; Prova ingresso di decodifica velocità per originare il ritardo del movimento della
; ruota di stampa
VLDC IN         0       ; Ingresso all'Accumulatore della Porta I/O 0
        RLC             ; Scorre il bit 7 nel Carry
        JNC     VLDC    ; Permane nell'anello se Carry è 0
; Alla fine del ritardo simula la commutazione on di FFE
        IN      1       ; Ingresso della Porta I/O 1
        ANI     DFH     ; RESET il bit 5
        ORI     10H     ; Pone bit 4
        OUT     1       ; Uscita del risultato
; Simula il ritardo di tempo di 2 ms di PW SETTLING
        MVI     A,0     ; Carica l'Accumulatore con 0
PWS     DCR     A       ; Decrementa A
        JNZ     PWS     ; Se A non è 0 seguita Decrementazione
```

Ci sono alcune variazioni interessanti in PW SETTLING e nel ciclo di ritardo di tempo di 2 millisecondi sopra illustrato.

Il ciclo di ritardo di tempo consiste di appena due istruzioni: l'istruzione DCR A, che decrementa i contenuti dell'Accumulatore e l'istruzione JNZ PSW che rimanda a DCR A se l'Accumulatore non contiene zero dopo che è stato eseguito il decremento. Queste due istruzioni si eseguono in 15 periodi di clock che aggiunge più di 7.5 microsecondi impiegando un clock di 500 nanosecondi.

Caricando inizialmente 0 nell'Accumulatore queste due istruzioni saranno eseguite 256 volte poichè il primo decremento dell'Accumulatore porterà da 0 ad FF_{16}.
Così **il tempo di ritardo totale è dato dalla seguente equazione:**

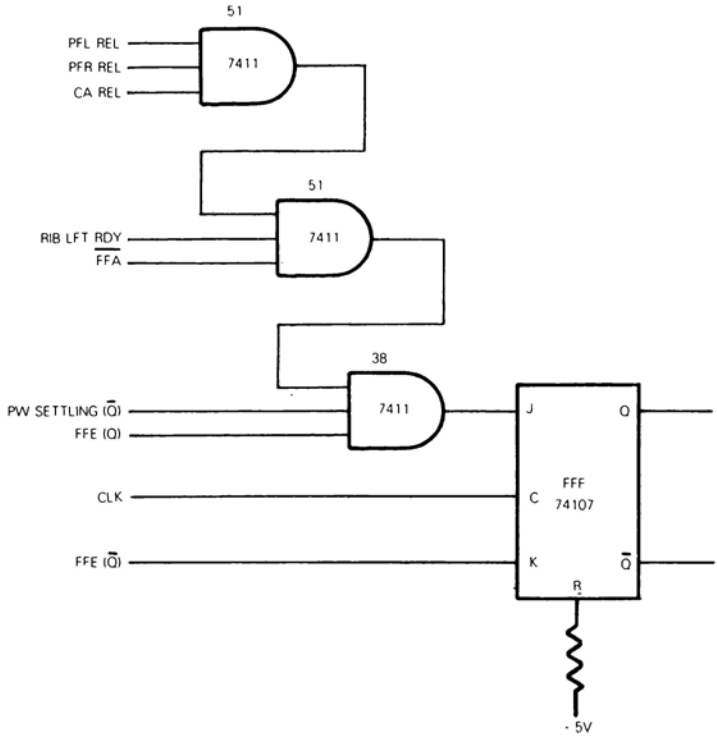


Dove 1923,5 microsecondi sono uguali a 1,9235 millisecondi.

Questo è abbastanza vicino a 2 millisecondi per gli scopi prefissati.

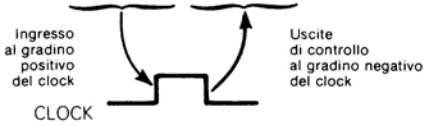
FLIP-FLOP FFF

Una volta emesso lo one-shot di PW SETTLING si è pronti per alimentare il martelletto di stampa. Il multivibratore 555 genera l'impulso di alimentazione del martelletto, ma è molto importante assicurarsi che il martelletto non venga alimentato mentre si sta muovendo qualsiasi parte del meccanismo di stampa o del carrello. Lo one-shot 555 è perciò fatto scattare dal flip-flop FFF che è commutato on da un ingresso J che è l'AND di molti segnali di sicurezza. Si isoli il flip-flop FFF e si esaminino i suoi ingressi.



Con il suo ingresso Clear (R) collegato a +5V il flip-flop FFF ha la seguente tabella di funzione:

Ingressi		Uscite	
J	K	Q	$\overline{Q}$
0	0	Non cambia	
1	0	1	0
0	1	0	1
1	1	Complementa	



Tra i cicli di stampa FFE è off cosicchè l'ingresso K di FFF è alto. L'ingresso J del flip-flop FFF sarà basso poichè l'uscita FFE (Q) sarà bassa ed FFE (Q) contribuisce ad FFF (J).

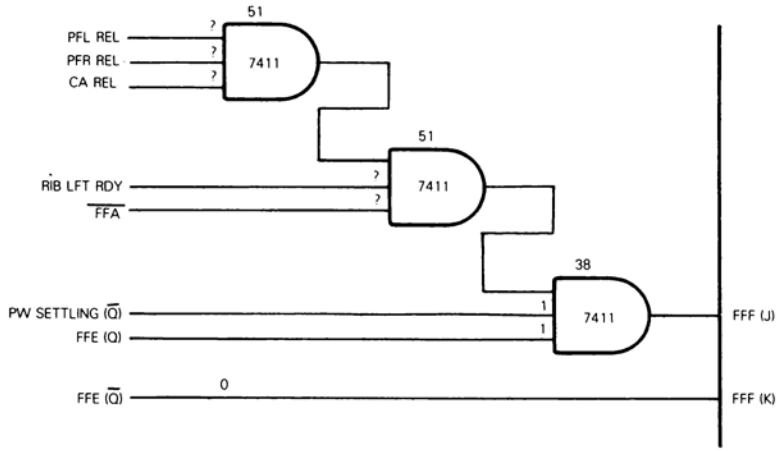
Perciò tra cicli di stampa il flip-flop è "off" poichè un ingresso J basso ed un ingresso R alto generano le uscite stazionarie Q=0 e $\overline{Q}$=1; questo è caratteristico di un flip-flop nella sua condizione off.

Ora quando FFE commuta on esso invia un K basso ad FFF. Mentre l'ingresso J è basso non si verificano cambiamenti. Non appena i sette segnali che contribuiscono ad FFF (J) sono tutti alti, il flip-flop FFF riceverà un ingresso J alto; questo farà commutare on il flip-flop FFF — la sua uscita Q sarà alta e $\overline{Q}$ bassa.

SIMULAZIONE DEL FLIP-FLOP FFF

Provenendo dalla simulazione di FFE si conosce che FFE (Q) ed FFE ($\overline{Q}$) hanno livelli corretti per fare commutare on FFF.

Si ricordi che dalla simulazione dello one-shot PW SETTLING, l'uscita one-shot $\overline{Q}$ deve essere alta:



Tutto questo è richiesto per verificare i cinque segnali di interlock rimanenti; non appena essi sono tutti alti, si simula il flip-flop FFF che commuta on. Questa è la sequenza d'istruzione:

```
; Impulso d'INIZIO MOVIMENTO DEL NASTRO al livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     8       ; Pone il bit 3 alto
        OUT     2       ; Uscita risultato
        ANI     F7H     ; Commuta off il bit 3
        OUT     2       ; Uscita risultato
; Prova ingresso decodifica di velocità per originare il ritardo del movimento della ruota
VLDC    IN      0       ; Ingresso della Porta I/O 0 all'Accumulatore
        RLC             ; Scorre il bit 7 nel Carry
        JNC     VLDC    ; Permane nell'anello se Carry è 0
; Alla fine del ritardo simula FFE che commuta on
        IN      1       ; Ingresso della Porta I/O 1
        ANI     DFH     ; RESET il bit 5
        ORI     10H     ; Pone il bit 4
        OUT     1       ; Uscita del risultato
; Simula il ritardo di tempo 2 ms di PW SETTLING
        MVI     A,0     ; Carica l'Accumulatore con 0
PWS     DCR     A       ; Decrementa A
        JNZ     PWS     ; Se A non è 0 lo ri-decrementa
```

```
; Simula il flip-flop FFF che commuta on
FFF  IN    0     ; Ingresso dei contenuti della Porta I/O 0 all'Accumulatore
     CMA         ; Complementa per la prova sui bit 1
     ANI   1FH   ; Isola i bit da 0 a 4
     JNZ   FFF   ; Se non c'è nessun bit 0 permane nell'anello
     IN    1     ; Ingresso della Porta I/O 1 nell'Accumulatore
     ORI   40H   ; Pone il bit 6 ad 1
     OUT   1     ; Uscita del risultato
```

Ora si dovrebbe essere in grado di comprendere le istruzioni aggiunte al programma.

Le prime quattro istruzioni caricano semplicemente i contenuti della Porta I/O 0 ed analizzano gli uni dei cinque bit di basso ordine. Finchè tutti i cinque bit non sono uguali ad uno, il programma rimane nell'anello delle quattro istruzioni che comincia con IN 0 e termina con JNZ FFF.

Quando i bit da 0 a 4 sono tutti uguali ad 1, l'istruzione CMA commuta a 0 tutti questi bit:

```
                                  Contenuti dell'Accumulatore
FFF   IN    0        XXX11111
      CMA            X̄X̄X̄00000
      ANI   1FH      00011111
                     ---------
                     00000000     Stato Zero = 1
      JNZ   FFF   Ritorno ad FFF solo se lo stato Zero è = 0
      IN    1     Continua qui se lo stato Zero è = 1
```

L'istruzione JNZ deflette l'esecuzione del programma ad FFF oppure consente la esecuzione della successiva istruzione sequenziale.

Le ultime tre istruzioni simulano il flip-flop FFF commutato on. Il bit 6 della Porta I/O 1 è stato assegnato ad FFF, quindi questo è il bit che deve essere posto ad 1.

Si può fare la modifica finale alla sequenza d'istruzione che pone correttamente lo stato del flip-flop tra cicli di stampa. Quindi si conclude con:

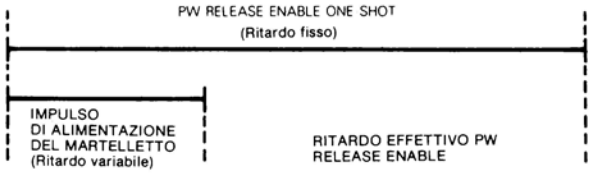
```
; Esecuzione del programma tra cicli di stampa
; Inizialmente pone a 0 il bit 6, 4, 3, 2 ed 1 ed a 1 il bit 0 della Porta I/O
     IN    1     ; Ingresso della Porta I/O 1 all'Accumulatore
     ORI   21H   ; Pone ad 1 i bit 5 e 0
     ANI   A1H   ; Ripristina a 0 i bit 6, 4, 3, 2 ed 1
     OUT   1     ; Ritorno risultato
; Prova per RETURN STROBE basso
L10  IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
     ANI   10H   ; Isola RETURN STROBE
     JZ    FFB   ; Se esso è 0 salta alla simulazione di FFB
```

Cosa succede quando il flip-flop FFF commuta on?

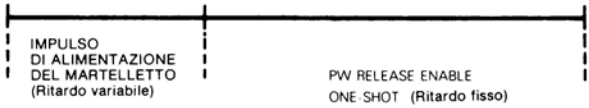
L'uscita FFF (Q) va al pin 9 del gate AND 37 di coordinate C7. Questa è una parte della logica che contribuisce al segnale PW REL. Comunque **la transizione dell'uscita FFF (Q) da basso ad alto non è significativa,** infatti l'altro ingresso al gate AND 37 è l'uscita FFD ($\overline{Q}$) che è attualmente bassa. L'uscita FFF (Q) è connessa al gate AND 37 per mantenere PW REL basso inizialmente nel ciclo di stampa quando FFD ($\overline{Q}$) è alta.

Le uscite FFF Q e $\overline{Q}$ contribuiscono agli ingressi J e K di FFC. FFF ($\overline{Q}$) è un contributo al gate AND 12 la cui uscita diventa l'ingresso FFC (J). L'altro contributo a questo gate AND è l'uscita del gate AND 37 di coordinate A4, che è costantemente alto durante il ciclo di stampa; perciò quando l'uscita FFF ($\overline{Q}$) diventa bassa anche l'ingresso FFC (J) lo diventa. L'ingresso K di FFC è l'uscita FFF (Q). **Quindi FFC commuterà off quando K diventa alto e questo non accade finchè FFF non commuta on.**

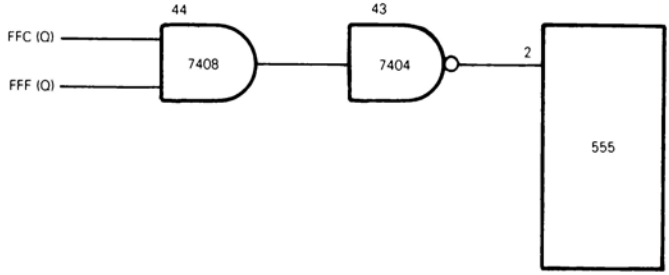
Comunque in questa simulazione si sta posticipando la commutazione on di FFC fino alla fine di HAMMBER PULSE. Questo perchè lo scopo della commutazione off di FFC è di far scattare lo one-shot PW RELEASE ENABLE che origina il ritardo di tempo necessario al martelletto per riposizionarsi. Così invece di impiegare ritardi paralleli:



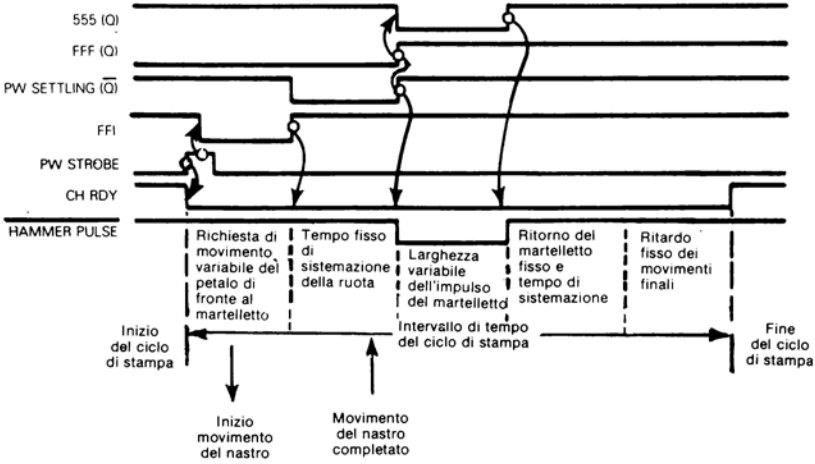
Si utilizzeranno ritardi seriali che si accordano meglio con la logica necessaria:



L'impulso di alimentazione del martelletto è generato dallo one-shot 555. Perciò lo one-shot 555 fornisce l'evento successivo della sequenza cronologica; esso è fatto scattare da una transizione da basso ad alto al pin 2. Questo pin d'ingresso è originato come segue:



Questa è la sequenza di eventi che deve essere simulata:



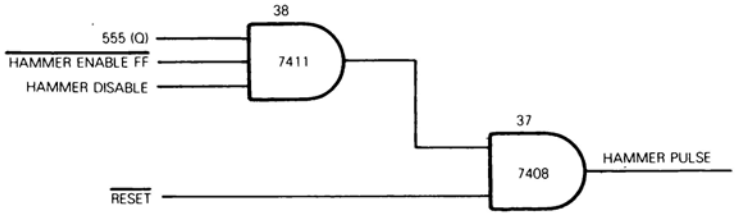
IL MULTIVIBRATORE 555

Si confronti il modo in cui il multivibratore 555 è stato collegato in Figura 3-1 con la descrizione del multivibratore fornita al Capitolo 2; si vedrà che **il flip-flop FFB commuta off il multivibratore tra cicli di stampa** inviando un reset basso al pin 4. **L'uscita FFF (Q) del flip-flop fa scattare il multivibratore,** come è stato appena descritto.

IMPULSO VARIABILE ONE-SHOT

La durata dell'impulso d'uscita one-shot è controllata dagli ingressi da H1 ad H6. Uno di questi ingressi sarà vero mentre tutti gli altri falsi; così il multivibratore, una volta fatto scattare, farà uscire uno one-shot che può avere un impulso alto con una delle sei durate possibili.

L'uscita one-shot del multivibratore 555 viene eventualmente invertita per trasformarla nell'impulso d'uscita del martelletto; comunque, per l'impulso d'uscita del martelletto sono necessari ingressi addizionali ai gates AND 37 e 38, localizzati alle coordinate B8 e C7 rispettivamente, che devono anche essere alti. Si può rappresentare come segue la logica dell'impulso del martelletto:



Si dovrà semplicemente provare l'ingresso HAMMER ENABLE FF prima della generazione di un'uscita HAMMER PULSE.

Occorre simulare l'interruttore HAMMERR DISABLE.

Invece RESET può essere ignorato poichè la logica RESET deve essere simulata tra i cicli di stampa.

SIMULAZIONE DEL MULTIVIBRATORE 555

La simulazione del multivibratore 555 consiste nella seguente sequenza logica:

1) **Si determini se sono soddisfatte le condizioni perchè un'uscita one-shot sia trasmessa come un'uscita HAMMER PULSE.**
2) **Si esaminino gli ingressi da H1 ad H6. Basandosi su questi ingressi, si origini uno dei sei possibili ritardi di tempo.**
3) **Se sono soddisfatte le condizioni per un'uscita HAMMER PULSE, si trasferisca l'uscita one-shot 555 in un'uscita HAMMER PULSE.**

Innanzitutto si consideri la logica di abilitazione di una uscita HAMMER PULSE.
La verifica della condizione HAMMER ENABLE FF è abbastanza semplice, essa è stata assegnata al pin 6 della Porta I/O 0.

LOGICA ESCLUSA DAL MICROCALCOLATORE

Come è possibile simulare la disabilitazione del martelletto se non esistono interruttori nei programmi in linguaggio assembly? Si potrebbe assegnare il pin rimanente — il pin 5 della Porta I/O 0 ad un segnale d'ingresso generato da un interruttore esterno. Potrebbe essere semplice posizionare questo interruttore

sulla linea di $\overline{\text{HAMMER ENABLE FF}}$ come segue:

Quindi si ignorerà l'interruttore di disabilitazione del martelletto e l'uscita dell'impulso del martelletto provvedendo che l'ingresso $\overline{\text{HAMMER ENABLE FF}}$ sia alto.

E per quanto riguarda le sei durate possibili dell'uscita del multivibratore 555? E' stato descritto al Capitolo 2 come si può originare un ritardo di tempo caricando un valore a 16-bit in due registri e quindi decrementando questi registri all'interno di un programma ad anello, permanendo in esso finchè il decremento non raggiunge lo zero. Si ripete qui l'anello di istruzioni:

```
      LXI   D,T16  ; Carica la costante di tempo in D ed E
LOOP  DCX   D      ; Decrementa D E
      MOV   A,D    ; Prova per 0 mediante l'operazione di OR
      OR    E      ; Tra i contenuti di D ed E attraverso l'Accumulatore
      JNZ   LOOP
```

La selezione di uno dei sei possibili ritardi di tempo è altrettanto semplice della selezione di una delle sei possibili costanti di tempo iniziali. Ora si può simulare il multivibratore 555 come segue:

```
      IN    1      ; Ingresso della Porta I/O 1 all'Accumulatore
      ORI   40H    ; Pone il bit 6 ad 1
      OUT   1      ; Uscita del risultato
; Prova HAMMER ENABLE FF
      IN    0      ; Ingresso della Porta I/O 0 all'Accumulatore
      ANI   40H    ; Isola il bit 6
      JZ    HPO    ; Se zero attraversa ponendo l'impulso del martelletto al
                   ; livello basso
; HAMMER ENABLE FF è alto, così l'impulso del martelletto
; Risulta al livello basso perciò pone a 0 il bit 2 della Porta I/O 3
      IN    2      ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI   FBH    ; Pone il bit 2 a 0
      OUT   2      ; Uscita risultato
; Calcola il ritardo di tempo
HPO   LXI   H,DELY ; Carica l'indirizzo base dei dati in HL
      LDA   H1 H6  ; Carica il selettore nell'Accumulatore
HP1   RRC          ; Ruota l'Accumulatore, pone Carry a 0
      INX   H      ; Incrementa di 2 i contenuti di HL
      INX   H
      JNC   HP1    ; Se RRC non ha spostato 1 nel Carry, ritorno
      MOV   D,M    ; Carica in D E la costante del tempo di ritardo a 16 bit
      INX   H
      MOV   E,M
TDLY  DCX   D      ; Esegue l'anello del ritardo di tempo
      MOV   A,D
      ORA   E
      JNZ   TDLY
; Impulso d'uscita martelletto ancora alto
      IN    2      ; Ingresso della Porta I/O 2 all'Accumulatore
      ORI   4      ; Pone ad 1 il bit 2
      OUT   2      ; Uscita risultato
```

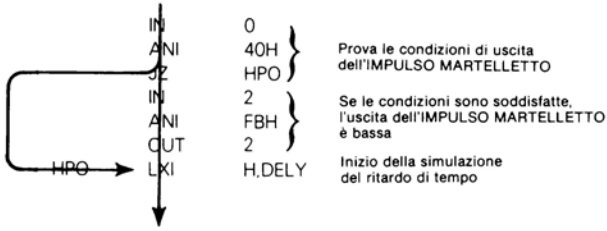
Rispetto agli altri dispositivi precedentemente simulati, il multivibratore 555 richiede molte istruzioni di simulazione. Mentre può sembrare ci sia molto da indagare la logica è invece abbastanza semplice; così esso sarà considerato una parte per volta.

SEGNALE ENABLE

Inizialmente si prova HAMMER ENABLE FF. HAMMER PULSE uscirà basso solo se HAMMER ENABLE FF è alto. Le tre istruzioni che provano lo stato di HAMMER ENABLE FF sono:

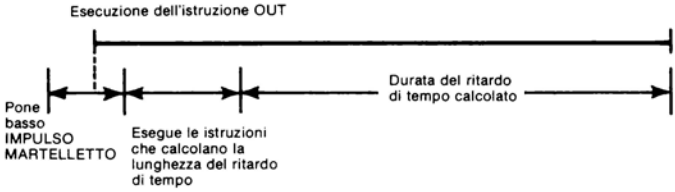
```
IN      0      ; Ingresso della Porta I/O 0 all'Accumulatore
ANI     40H    ; Isola il bit 6
JNZ     HPO    ; Se 0, attraversa ponendo HAMMER PULSE basso
```

Ci sono due aspetti di queste tre istruzioni che necessitano di essere spiegati. Primo, la logica deve essere realizzata. Si sta determinando se sono soddisfatte le condizioni per cui HAMMER PULSE risulti basso. Se le condizioni sono soddisfatte allora, immediatamente, HAMMER PULSE sarà basso; se le condizioni non sono soddisfatte l'istruzione JZ HPO fa aggirare la sequenza d'istruzione che fa uscire basso HAMMER PULSE:

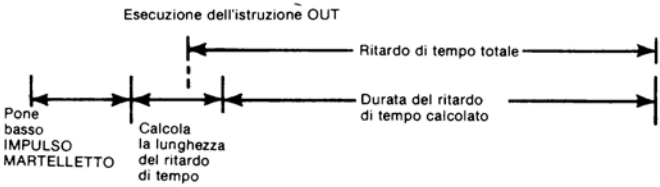


SEQUENZA DI EVENTO

L'impulso di uscita del martelletto diventa basso prima di iniziare il calcolo della durata dell'impulso di tempo; perchè è così? La ragione è di risparmiare tempo. Le istruzioni che calcolano la lunghezza del ritardo di tempo possono essere eseguite dall'inizio del ritardo di tempo.



Il ritardo di tempo potrebbe essere calcolato così semplicemente, quindi si pone basso HAMMER PULSE, quindi si esegue il ritardo di tempo, gli eventi potrebbero cronologicamente verificarsi come segue:



La sovrapposizione degli eventi nel tempo aumenta molto la sensibilità.

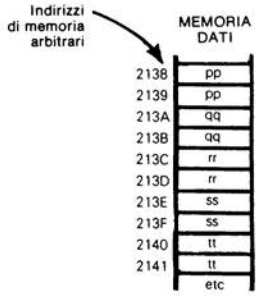
Il metodo effettivamente impiegato per calcolare il ritardo di tempo richiede una piccola spiegazione. **Alla fine del programma ci saranno 12 bytes di memoria nei quali saranno immagazzinate 6 costanti a 16-bit.**

Ecco come lavora il programma sorgente:

```
HP0   LXI   H,DELY  ; Carica l'indirizzo del primo ritardo di tempo in HL
      LDA   H1 H6   ; Carica il selettore nell'Accumulatore
HP1   RRC           ; Ruota l'Accumulatore, pone Carry a 0
      INX   H       ; Incrementa di 2 i contenuti di H L
      INX   H
      JNC   HP1     ; Se RRC non fa scorrere 1 nel Carry, ritorno
      MOV   D,M     ; Carica in D E la costante del tempo di ritardo a 16-bit
      INX   H
      MOV   E,M
TDLY  DCX   D       ; Esegue l'anello del ritardo di tempo
      MOV   A,D
      ORA   E
      JNZ   TDLY
: Impulso d'uscita martelletto ancora alto
      IN    2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ORI   4       ; Pone il bit 2 ad 1
      OUT   2       ; Uscita risultato
      —
      —
      —
      ORG   DELY+2
      pppp          ; Ritardo di tempo H1
      qqqq          ; Ritardo di tempo H2
      rrrr          ; Ritardo di tempo H3
      ssss          ; Ritardo di tempo H4
      tttt          ; Ritardo di tempo H5
      uuuu          ; Ritardo di tempo H6
```

Le lettere p, q, r, s, ed u sono state impiegate per rappresentare valori esadecimali. I sei ritardi di tempo possono essere rappresentati da qualunque valore numerico da 0000_{16} ad $FFFF_{16}$.

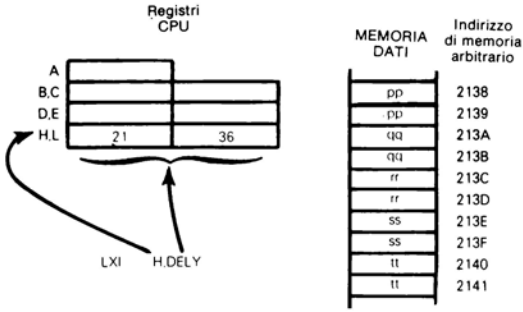
L'indirizzo del primo byte di memoria nel quale viene immagazzinato il primo ritardo di tempo è dato dall'espressione DELY+2. Si supponga che questa locazione di memoria sia la 2318:



DELY è la label alla quale deve essere assegnato il valore 2318. Questa assegnazione viene fatta impiegando un direttivo di uguaglianza che dovrebbe apparire all'inizio del programma come segue:

```
      DELY    EQU    1316 H
```

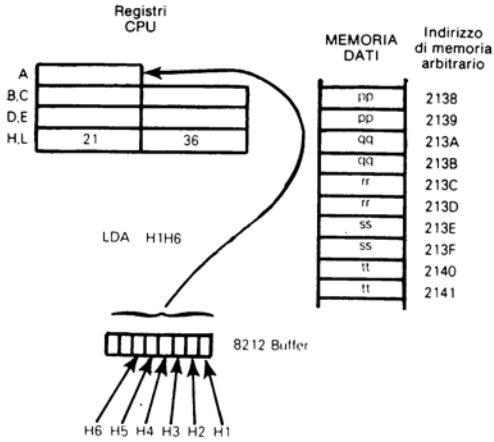
Ora si comincia il calcolo del ritardo di tempo caricando l'indirizzo DELY nei registri H ed L. Si assume che la label DELY abbia il valore 2316H, come sopra illustrato. Dopo che è stata eseguita l'istruzione LXI H, DELY la situazione è la seguente:



L'istruzione successiva, LDA H1H6, carica nell'Accumulatore i contenuti del buffer 8212 ad 8-bit. L'indirizzo di memoria che permette al buffer di autoselezionarsi è rappresentato dalla label H1H6. Si supponga che questo indirizzo di memoria sia $FFFF_{16}$; allora H1H6 dovrebbe avere assegnato il valore $FFFF_{16}$ mediante un direttivo di uguaglianza all'inizio del programma come segue:

```
DELY    EQU     2316H
H1H6    EQU     FFFFH
```

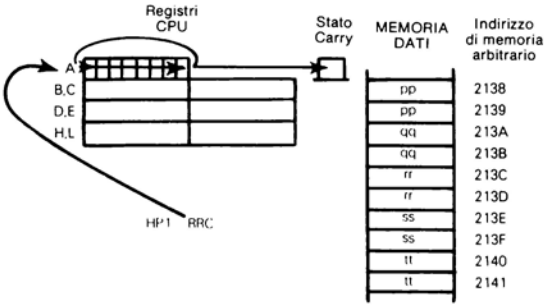
Dalla discussione sui segnali d'ingresso si ricordi che solo uno dei segnali dei sei ingressi da H1 ad H6 sarà alto e tutti gli altri cinque segnali sono bassi. Perciò **dopo che è stata eseguita l'istruzione LDA, l'Accumulatore conterrà un 1 in uno dei sei bit di basso ordine:**



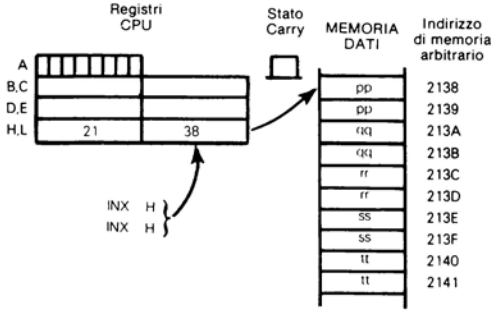
CALCOLO DELL'INDIRIZZO DELLA MEMORIA DATI

Si può calcolare l'indirizzo del ritardo di tempo richiesto aggiungendo 2 ai contenuti H ed L un numero di volte dato dalla posizione del bit 1 nell'Accumulatore. Questo può essere illustrato come segue:

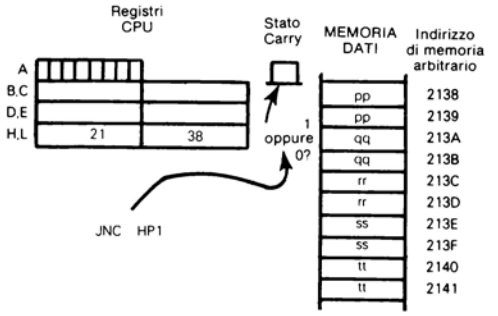
① Scorre i contenuti dell'Accumulatore a destra di un bit e nel Carry:



② Somma 2 ad H L:



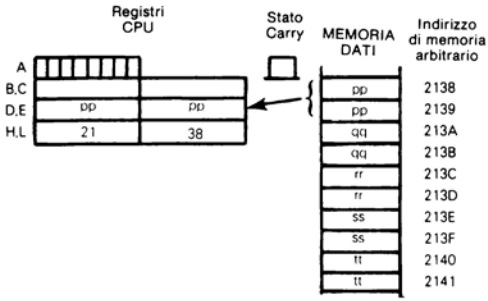
③ Se lo stato Carry non è 1 ritorna al punto 1, altrimenti HL contiene l'indirizzo corretto:



La logica per fare la somma dell'indirizzo richiesto è fornita da queste quattro istruzioni:

```
HP1  RRC          ; Ruota l'Accumulatore, pone Carry a 0
     INX   H      ; Incrementa HL di 2
     INX   H
     JNC   HP1    ; Se RRC non ha spostato 1 nel Carry, ritorno
```

Ora che il ritardo di tempo corretto è indirizzato dai registri H ed L, si carica l'appropriata costante di ritardo a 16-bit in D ed E. Si supponga che H1 fosse il segnale di ingresso alto, il risultato è il seguente:



L'indirizzo selezionato pppp è mosso dai registri D ed E mediante le tre istruzioni:

```
     MOV   D,M    ; Muove i contenuti del byte 2138 al Registro D
     INX   H      ; Indirizzo del byte 2139
     MOV   E,M    ; Muove i contenuti del byte 2139 al Registro E
```

Il ritardo di tempo effettivo è originato da questo anello di istruzione, che è stato descritto al Capitolo 2:

```
TDLY DCX   D      ; Decrementa il contatore di ritardo
     MOV   A,D    ; Prova per 0 in D e mediante OR di D
     ORA   E      ; Con E nell'Accumulatore
     JNZ   TDLY   ; Ritorno se non zero
```

Le ultime tre istruzioni fanno uscire alto l'uscita HAMMER PULSE, senza eseguire nessuna prova per vedere se HAMMER PULSE era basso. Questa logica opererà così poichè, uscendo alto HAMMER PULSE, non ci saranno effetti apprezzabili perchè è già alto. In queste condizioni il tempo richiesto per eseguire le ultime tre istruzioni è semplicemente sprecato. Poichè sarebbero necessarie tre istruzioni per controllare se HAMMER PULSE è stato posto al livello basso, lo spreco è giustificato.

CALCOLO DEL RITARDO DI TEMPO

Si consideri il tempo impiegato per calcolare il ritardo di tempo. I tempi di esecuzione per istruzioni importanti sono elencati di seguito:

Cicli		Istruzione		
		IN	2	
		ANI	FBH	
		OUT	2	← Impulso basso del martelletto comincia qui
10	HP0	LXI	H,DELY	
13		LDA	H1H6	
4	HP1	RRC		Queste quattro istruzioni saranno eseguite da 1 a 6 volte. In questo anello ci sono 26 cicli
5		INX	H	
5		INX	H	
10		JNC	HP1	
7		MOV	D,M	
5		INX	H	
7		MOV	E,M	
5	TDLY	DCX	D	
5		MOV	A,D	
4		ORA	E	
10		JNZ	TDLY	
10		IN	2	
7		ORI	4	
10		OUT	2	← Impulso basso del martelletto termina qui
117				

Assumendo un clock di 500 nanosecondi, il tempo di esecuzione è dato dall'espressione:

(46,5 + 12 x N) microsecondi

dove N è un numero tra 1 per l'impulso più corto e 6 per l'impulso più lungo. **Perciò i tempi di esecuzione andranno da 58,5 a 118,5 microsecondi. Questi tempi devono essere sottratti dai ritardi successivamente generati.** Per esempio si supponga che H1 alto richieda al 555 di uscire con un segnale one-shot che è alto per 1,66 millisecondi (approssimativamente) e, dopo un ritardo di 1,6 millisecondi, aggiunge un tempo di assestamento di 58,5 microsecondi.

IL FLIP-FLOP PW RELEASE ENABLE

Non appena l'uscita one-shot del 555 diviene ancora bassa, il flip-flop FFC viene simulato che commuta off. Quando FFC commuta off, la sua uscita $\overline{Q}$ compie una transizione da basso ad alto e questo fa scattare lo one-shot PW RELEASE ENABLE. Questo è uno one-shot 74121 identificato da 36 alle coordinate approssimativamente E2. Lo scopo di questo one-shot e di permettere il tempo al martelletto di riposizionarsi prima che sia fatto qualsiasi tentativo di riposizionamento della ruota di stampa. **Questo è stato illustrato come il ritardo di tempo di riposizionamento fissato dopo il ritorno del martelletto.**

SIMULAZIONE DEL FLIP-FLOP PW RELEASE ENABLE

RITARDO DI TEMPO

Questa è, in realtà, una simulazione in due parti; prima si deve simulare la commutazione off del flip-flop FFC, poi si deve eseguire un ritardo di tempo appropriato. Un ritardo di tempo di 3 millisecondi è sufficiente. Le istruzioni che commutano off il flip-flop FFC saranno eseguite all'interno di un ritardo di tempo di 3 millisecondi. Il ritardo di tempo calcolato sarà perciò un po' minore di 3 millisecondi. Ecco una sequenza d'istruzione appropriata.

```
JNC   HP1   ; Se RRC non sposta 1 nel Carry si ha il ritorno
MOV   D,M   ; Carica in D E la costante del tempo di ritardo a 16-bit
INX   H
MOV   E,M
```

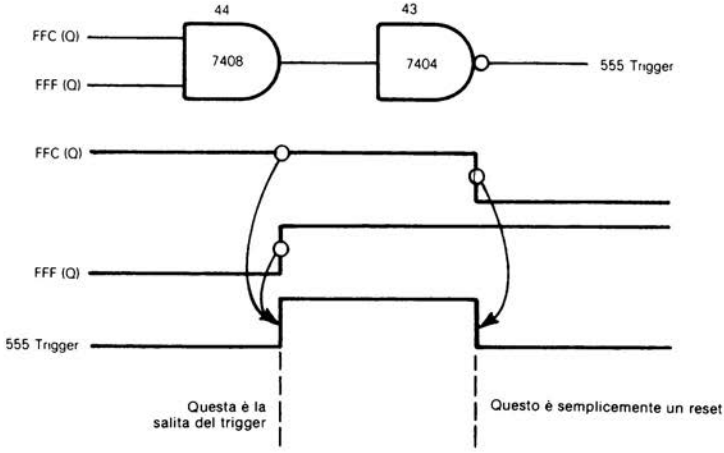
```
TDLY DCX    D       ; Esegue l'anello del ritardo di tempo
     MOV    A,D
     ORA    E
     JNZ    TDLY
; Impulso d'uscita del martelletto ancora alto
     IN     2       ; Ingresso della Porta I/O 2 all'Accumulatore
     ORI    4       ; Pone ad 1 il bit 2
     OUT    2       ; Uscita risultato
; Commuta off il flip-flop FFC
     IN     1       ; Ingresso della Porta I/O 1 all'Accumulatore
     ANI    FBH     ; Pone a 0 il bit 2
     OUT    1       ; Uscita risultato
; Esegue un ritardo di tempo di 3 millisecondi
     LXI    D,F7H   ; Carica la costante di tempo in D, E
PWR1 DCX    D       ; Decrementa D, E
     MOV    A,D     ; Prova per risultato zero
     ORA    E
     JNZ    PWR1    ; Decrementa se non zero
```

La costante di tempo F7H è uguale a 247_{10}. Le quattro istruzioni precedenti l'anello vengono eseguite in 34 microsecondi e le quattro istruzioni dell'anello di ritardo in 12 microsecondi. Perciò il tempo di ritardo totale è dato dall'equazione:

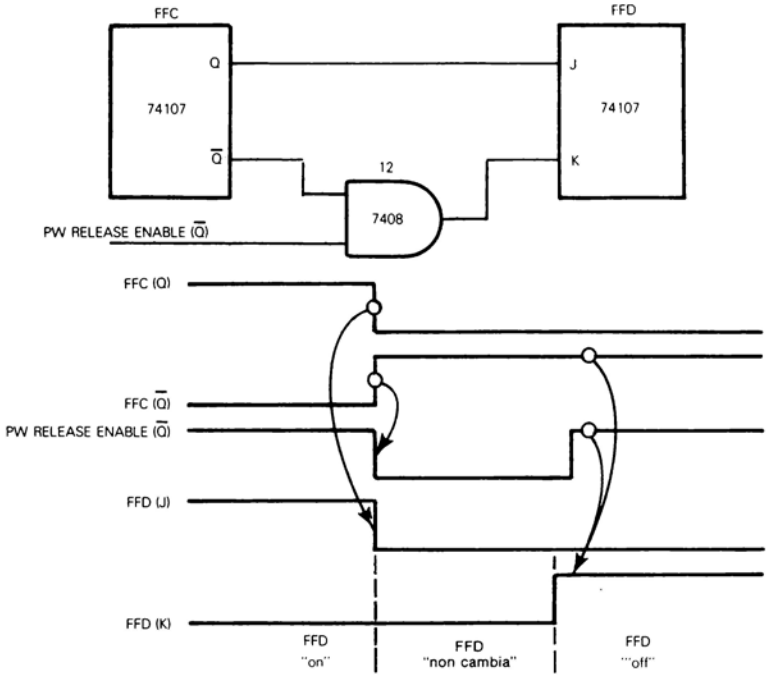
$$247 \times 12 + 34 = 2998 \text{ microsecondi}$$

In realtà il ritardo di 3 millisecondi non è un numero critico; 2,5 o 3,5 millisecondi probabilmente potrebbero comportarsi allo stesso modo cosicchè la scelta di 34 microsecondi non è significativa in questo caso. Tuttavia, nelle applicazioni successive, la durata di un ritardo di tempo può essere molto critica; allora le considerazioni di timing precedentemente discusse saranno molto significative.

Allo scopo di determinare cosa succede alla fine del ritardo di tempo PW RELEASE si devono osservare le uscite Q e $\overline{Q}$ di FFC. L'uscita Q è collegata al gate AND DELL'IMPULSO DI INIZIO MOVIMENTO DEL NASTRO ed alla logica del trigger one-shot 555; in nessuno dei due casi la transizione da alto a basso di Q ha qualche effetto. Il segnale dell'impulso di INIZIO MOVIMENTO DEL NASTRO è già basso e lo one-shot 555 è fatto scattare da una transizione di Q da basso ad alto. La transizione da alto a basso fa semplicemente cadere il segnale di trigger ad un livello basso che non richiede simulazione:



L'uscita FFC ($\overline{Q}$) subisce l'AND con lo one-shot PW RELEASE ENABLE $\overline{Q}$ in modo da generare l'ingresso FFD (K). L'ingresso FFD (J) proviene direttamente da FFC (Q), perciò **non appena lo one-shot PW RELEASE ENABLE diventa ancora alto, FFD riceverà un ingresso J basso ed un K alto:**



L'ingresso J basso e K alto fanno commutare off il flip-flop FFD; questo fa scattare lo one-shot PW READY ENABLE.

SIMULAZIONE DELLO ONE-SHOT PW READY ENABLE

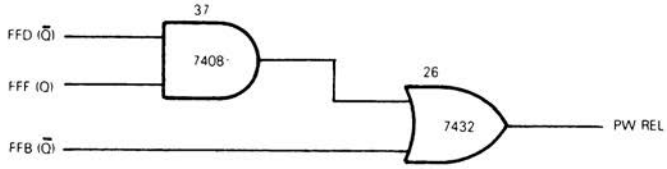
La logica associata con questo one-shot è in maggior parte identica allo one-shot PW RELEASE ENABLE. La commutazione off di FFD origina una transizione dell'uscita $\overline{Q}$ da basso ad alto che fa scattare lo one-shot PW READY ENABLE.

Si deve ora simulare un ritardo di tempo di 2 millisecondi; a parte questo la sequenza d'istruzione successiva è pressochè identica a quella della simulazione dallo one-shot PW RELEASE ENABLE e può essere illustrata come segue:

```
      JNC   HP1    ; Se RRC non scorre 1 nel Carry, ritorno
      MOV   D,M    ; Carica in D E la costante del tempo di ritardo a 16-bit
      INX   H
      MOV   E,M
TDLY  DCX   D      ; Esegue l'anello del ritardo di tempo
      MOV   A,D
      ORA   E
      JNZ   TDLY
```

```
; Impulso d'uscita del martelletto ancora al livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     4       ; Pone ad 1 il bit 2
        OUT     2       ; Uscita risultato
; Commuta off il flip-flop FFC
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     FBH     ; Pone a 0 il bit 2
        OUT     1       ; Uscita risultato
; Esegue un ritardo di tempo di 3 millisecondi
        LXI     D,F7H   ; Carica la costante di tempo in D, E
PWR1    DCX     D       ; Decrementa D, E
        MOV     A,D     ; Prova per risultato zero
        ORA     E
        JNZ     PWR1    ; Ri-decrementa se non zero
; Commuta off il flip-flop FFD
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     F7H     ; Pone a 0 il bit 3
        OUT     1       ; Uscita risultati
; Esegue un ritardo di tempo di 2 millisecondi
        MVI     A,83H   ; Carica la costante di tempo nell'Accumulatore
PWR2    DCR     A       ; Decrementa l'Accumulatore
        JNZ     PWR2    ; Ri-decrementa se non zero
```

Quando FFD commuta off l'uscita PW REL diventa ancora alta. Ecco la logica che origina PW REL:



FFB ($\overline{Q}$) è ancora basso in questo istante. Però FFD ($\overline{Q}$) ed FFF ($\overline{Q}$) sono entrambi alti cosicchè il gate AND 37 fa uscire un livello alto che, passando attraverso il gate OR 26, pone alto PW REL.

Queste istruzioni pongono alto PW REL:

```
        JNC     HP1     ; Ritorno se RRC non fa scorrere 1 nel Carry
        MOV     D,M     ; Carica in D E la costante del ritardo di tempo a 16-bit
        INX     H
        MOV     E,M
TDLY    DCX     D       ; Esegue l'anello del tempo di ritardo
        MOV     A,D
        ORA     E
        JNZ     TDLY
; Impulso d'uscita del martelletto ancora al livello alto
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     4       ; Pone ad 1 il bit 2
        OUT     2       ; Uscita risultato
; Commuta off il flip-flop FFC
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     FBH     ; Pone a 0 il bit 2
        OUT     1       ; Uscita risultati
```

```
; Esegue un ritardo di tempo di 3 millisecondi
        LXI     D,F7H   ; Carica la costante di tempo in D, E
PWR1    DCX     D       ; Decrementa D, E
        MOV     A,D     ; Prova per risultato zero
        ORA     E
        JNZ     PWR1    ; Ri-decrementa se non zero
; Commuta off il flip-flop FFD
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     F7H     ; Pone a 0 il bit 3
        OUT     1       ; Uscita risultato
; Esegue un ritardo di tempo di 2 millisecondi
        MVI     A,83H   ; Carica la costante di tempo nell'Accumulatore
PWR2    DCR     A       ; Decrementa l'Accumulatore
        JNZ     PWR2    ; Ri-decrementa se non zero
; Pone alto PW REL
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     1       ; Pone ad 1 il bit 0
        OUT     2       ; Uscita risultato
```

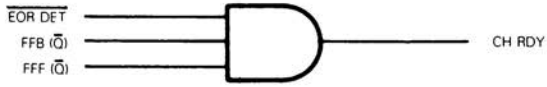
Ora l'intero ciclo di stampa termina velocemente. Le uscite Q e $\overline{Q}$ del flip-flop FFD divengono gli ingressi J e K di FFE. Q subisce prima l'AND con FFI che in questo istante è costantemente alto; perciò all'istante in cui FFD commuta OFF, FFE riceve un ingresso J basso.

L'ingresso FFE (K) non diventa alto finchè non termina lo one-shot PW READY ENABLE poichè l'uscita $\overline{Q}$ di PW READY ENABLE subisce l'AND con $\overline{Q}$ da FFD in modo da generare FFE (K).

La commutazione off di FFE è ora l'evento cronologicamente successivo.

La commutazione off di FFE, a sua volta, origina la commutazione off degli FFB ed FFF. FFB è commutato off perchè i suoi ingressi J e K sono collegati direttamente alle uscite Q e $\overline{Q}$ di FFE.

Una volta commutati off FFB ed FFF sono soddisfatte tutte le condizioni perchè CH RDY sia ancora commutato alto fornendo $\overline{\text{EOR DET}}$ se non viene segnalata la fine del nastro:



Così si può concludere la simulazione come segue:

```
        JNZ     PWR1    ; Ri-decrementa se non zero
; Commuta off il flip-flop FFD
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     F7H     ; Pone a 0 il bit 3
        OUT     1       ; Uscita risultato
; Esegue un ritardo di tempo di 2 millisecondi
        MVI     A,83H   ; Carica la costante di tempo nell'Accumulatore
PWR2    DCR     A       ; Decrementa l'Accumulatore
        JNZ     PWR2    ; Ri-decrementa se non zero
; Pone alto PW REL
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     1       ; Pone ad 1 il bit 0
        OUT     2       ; Uscita risultato
; Commuta off i flip-flop FFB, FFE ed FFF
        IN      1       ; Ingresso della Porta I/O 1 all'Accumulatore
        ANI     ADH     ; RESET a 0 i bit 1, 4 e 6
        ORI     20H     ; Pone ad 1 il bit 5
        OUT     1       ; Uscita risultato
; Pone alto CH RDY
        IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ORI     2       ; Pone ad 1 il bit 1
        OUT     2       ; Uscita risultato
; Divide per la prova di fine valida del ciclo di stampa
        JMP     LOP1
```

SOMMARIO DELLA SIMULAZIONE

Il programma della simulazione completa sviluppato in questo capitolo è fornito in Figura 3-3.

Si può concludere che non è possibile una simulazione assolutamente esatta, uno ad uno della logica digitale impiegando le istruzioni in linguaggio assembly all'interno di un sistema a microcalcolatore; ma allora il sistema a microcalcolatore non è particolarmente auspicabile in questo caso.

Per i lettori non progettisti di logica digitale probabilmente esisterà una confusione tra le varie combinazioni di segnale richieste all'interno della logica di Figura 3-1. Un grande traguardo di ciò è che non c'è nulla da fare sulle richieste finali della stampante Qume; piuttosto esso riflette una realizzazione logica interna del progetto logico, diretta ad assicurare delle sequenze di segnale esterne opportune in tutte le circostanze concepibili.

Per un progettista logico i cambiamenti che realizzerebbero le richieste specifiche della stampante Qume si collegano in modo completamente diverso; poi la realizzazione potrebbe essere difficoltosa.

Il punto importante da fissare in mente è che la logica digitale contiene molte sottigliezze specifiche dei dispositivi a logica discreta. Queste sottigliezze non sono collegate alle specifiche della realizzazione globale.

Ora il linguaggio assembly ha un proprio insieme di sottigliezze ed anche queste non hanno nulla a che fare con la realizzazione finale; piuttosto essere aiutano a rendere più effettivo l'impiego di singole o sequenze d'istruzione.

Quindi non dovrebbe sorprendere che una duplicazione esatta della logica digitale, impiegando il linguaggio assembly, non sia fattibile né desiderabile. Così ci si muoverà dalla logica digitale alla trattazione di un problema dal punto di vista della programmazione.

IL LINGUAGGIO ASSEMBLY RISPETTO ALLA LOGICA DIGITALE

La differenza principale tra la logica digitale ed il linguaggio assembly è che il linguaggio assembly tratta gli eventi cronologicamente mentre la logica digitale segrega la logica in nodi funzionali. Così un dispositivo logico può essere responsabile di un numero di eventi verificatisi in tempi diversi durante qualsiasi ciclo logico; nella traduzione in un programma in linguaggio assembly ogni evento diventa una sequenza d'istruzione separata.

Nella Figura 3-1 per esempio, il ciclo di stampa viene avviato dalla commutazione on di una cascata di flip-flop e termina con la stessa commutazione off dei flip-flop. In molti casi la commutazione on di un flip-flop fa scattare un evento, mentre la commutazione off dello stesso flip-flop fa scattare un evento completamente diverso. All'interno di un programma in linguaggio assembly i due eventi non avranno nulla in comune. Ogni evento sarà rappresentato da una sequenza d'istruzione completamente indipendente, localizzata in parti sostanzialmente diverse del programma.

L'altra maggiore differenza tra la logica digitale ed il programma in linguaggio assembly è il concetto di timing. All'interno della logica digitale sincrona, come illustrato in Figura 3-1, il timing è confinato ai segnali di clock e la richiesta di assenza di interazione di segnale. All'interno di un programma in linguaggio assembly il timing deriva direttamente dalla sequenza in cui vengono eseguite le istruzioni. Inoltre, mentre i componenti di un circuito digitale possono commutare e funzionare in parallelo, all'interno di un programma in linguaggio assembly tutto si verifica in modo seriale.

Ora il concetto chiave da afferrare da questo capitolo è che non c'è nulla di più giusto nella logica digitale di realizzare ogni cosa. Il fatto che non si è stati in grado di duplicare esattamente la logica digitale impiegando le istruzioni in linguaggio assembly non significa che il linguaggio assembly è in qualche modo inferiore; questo significa semplicemente che il linguaggio assembly esegue il lavoro in un modo diverso.

In questo capitolo è stato tracciato un parallelismo diretto tra il linguaggio assembly e la logica digitale; ora si abbandonerà qualsiasi tentativo a favore della logica digitale. Precisamente al Capitolo 4 la logica illustrata in Figura 3-1 sarà ri-simulata ma dal punto di vista del programmatore.

```
; Assegna le locazioni alla tabella di conteggio del ritardo
; E le linee di selezione della durata del tempo
DELY  EQU     xxxx
H1H6  EQU     yyyy
; Prova per fine valida del ciclo di stampa
LOP1  IN      2       ; Ingresso dei contenuti della Porta I/O 2 all'Accumulatore
      RLC             ; Sposta il bit 7 nel Carry
      JNC     LOP1    ; Se nel Carry c'è zero, permane nel ciclo di stampa
; Esecuzione del programma tra cicli di stampa
; Inizialmente pone a 0 i bit 6, 4, 3, 2 ed 1, pone ad 1 i bit 5 e 0
      IN      1       ; Ingresso all'Accumulatore della Porta I/O 1
      ORI     21H     ; Pone ad 1 i bit 5 e 0
      ANI     A1H     ; RESET a 0 dei bit 6, 4, 3, 2 ed 1
      OUT     1       ; Ritorno risultati
; Prova per RETURN STROBE basso
L10   IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
      ANI     10H     ; Isola RETURN STROBE
      JZ      FFB     ; Se è 0, salta alla simulazione di FFB
; Simulazione di FFA e della logica associata
; Carica nell'Accumulatore i contenuti della Porta I/O 2 ed
; Isola i bit 1, 5 e 6 per CH RDY, PW STROBE
; E RESET, rispettivamente
      IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
      ANI     62H     ; Isola i bit 6, 5 ed 1
      CPI     22H     ; Se RESET=0, CH RDY=1 e PW STROBE=1
      JNZ     L10     ; Inizia un nuovo ciclo di stampa altrimenti ritorna ad L10
      IN      1       ; Per iniziare un nuovo ciclo di stampa, pone
      ANI     FEH     ; A 0 il bit 0 della Porta I/O 1
      OUT     1
; La sequenza del nuovo ciclo di stampa inizia qui
; Simula il flip-flop FFB che commuta on
FFB   IN      1       ; Carica nell'Accumulatore la Porta I/O 1
      ANI     FDH     ; Ripristina a 0 il bit 1
      OUT     1       ; Ri-immagazzina il risultato
; Simula il gate AND 7411 che commuta basso CH RDY
; Inoltre simula il gate OR 7432 che commuta basso PW REL
      IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
      ANI     FCH     ; RESET a 0 dei bit 0 ed 1
      OUT     2       ; Ri-immagazzina il risultato
; CH RDY basso commuta off FFA, pone ad 1 il bit 0 della Porta I/O 1
; Inoltre simula FFC ed FFD che commutano on pone ad 1 i bit 3 e 2 della Porta I/O 1
      IN      1       ; Carica nell'Accumulatore la Porta I/O 1
      ORI     0DH     ; Pone ad 1 i bit 3, 2 e 0
      OUT     1       , Ri-immagazzina il risultato
; Impulso alto d'INIZIO MOVIMENTO DEL NASTRO
      IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
      ORI     8       ; Pone alto il bit 3
      OUT     2       ; Uscita risultato
      ANI     F7H     ; Pone off il bit 3
      OUT     2       ; Uscita risultato
```

Figura 3-3. Il programma di simulazione completa (segue)

```
; Prova ingresso decodifica di velocità per originare il ritardo del movimento della
; ruota di stampa
VLDC IN      0       ; Ingresso all'Accumulatore della Porta I/O 0
     RLC             ; Sposta il bit 7 nel Carry
     JNC     VLDC    ; Permane nell'anello se Carry è 0
; Alla fine del ritardo simula FFE che commuta on
     IN      1       ; Ingresso della Porta I/O 1
     ANI     DFH     ; RESET del bit 5
     ORI     10H     ; Pone bit 4 ad 1
     OUT     1       ; Uscita del risultato
; Simula il ritardo di tempo di 2ms di PW SETTLING
     MVI     A,0     ; Carica l'Accumulatore con 0
PWS  DCR     A       ; Decrementa A
     JNZ     PWS     ; Se A non è decrementato a 0, ridecrementa
; Simula il flip-flop FFF che commuta on
FFF  IN      0       ; Ingresso all'Accumulatore dei contenuti della Porta I/O 0
     CMA             ; Complementa per provare tutti i bit 1
     ANI     IFH     ; Isola i bit da 0 a 4
     JNZ     FFF     ; Se c'era qualsiasi bit 0, permane nell'anello
     IN      1       ; Ingresso all'Accumulatore della Porta I/O 1
     ORI     40H     ; Pone ad 1 il bit 6
     OUT     1       ; Uscita del risultato
  Prova HAMMER ENABLE FF
     IN      0       ; Ingresso all'Accumulatore della Porta I/O 0
     ANI     40H     ; Isola il bit 6
     JNZ     HP0     ; Se 0 scavalca ponendo HAMMER PULSE basso
; HAMMER ENABLE FF è alto cosicchè HAMMER PULSE
; Deve uscire basso, perciò pone a 0 il bit 2 della Porta I/O 3
     IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
     ANI     FBH     ; Pone a 0 il bit 2
     OUT     2       ; Uscita risultato
; Calcola il ritardo di tempo
HP0  LXI     H,DELY  ; Carica la base di indirizzo dati in HL
     LDA     H1H6    ; Carica nell'Accumulatore il selettore
HP1  RRC             ; Ruota l'Accumulatore, pone Carry a 0
     INX     H       ; Incrementa di 2 i contenuti di HL
     INX     H
     JNC     HP1     ; Se RRC non sposta 1 nel Carry ritorno
     MOV     D,M     ; Carica la costante di ritardo di tempo a 16-bit in DE
     INX     H
     MOV     E,M
TDLY DCX     D       ; Esegue l'anello del ritardo di tempo
     MOV     A,D
     ORA     E
     JNZ     TDLY
; Uscita impulso martelletto ancora alta
     IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
     ORI     4       ; Pone ad 1 il bit 2
     OUT     2       ; Uscita risultato
; Commuta off il flip-flop FFC
     IN      1       ; Ingresso all'Accumulatore della Porta I/O 1
     ANI     FBH     ; Pone a 0 il bit 2
     OUT     1       ; Uscita risultati
```

Figura 3-3. Il programma di simulazione completa (segue)

```
; Esegue un ritardo di tempo di 3 millisecondi
        LXI     D,F7H   ; Carica la costante di tempo in D, E
PWR1    DCX     D       ; Decrementa D, E
        MOV     A,D     ; Prova per risultato zero
        ORA     E
        JNZ     PWR1    ; Ri-decrementa se non zero
; Commuta off il flip-flop FFD
        IN      1       ; Ingresso all'Accumulatore della Porta I/O 1
        ANI     F7H     ; Pone a 0 il bit 3
        OUT     1       ; Uscita risultati
; Esegue un ritardo di tempo di 2 millisecondi
        MVI     A,83H   ; Carica la costante di tempo nell'Accumulatore
PWR2    DCR     A       ; Decrementa l'Accumulatore
        JNZ     PWR2    ; Ri-decrementa se non zero
; Pone alto PW REL
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     1       ; Pone ad 1 il bit 0
        OUT     2       ; Uscita risultato
; Commuta off i flip-flop FFB, FFE ed FFF
        IN      1       ; Ingresso all'Accumulatore della Porta I/O 1
        ANI     ADH     ; Ripristina a 0 i bit 1, 4 e 6
        ORI     20H     ; Pone ad 1 il bit 6
        OUT     1       ; Uscita risultati
; Pone alto CH RDY
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     2       ; Pone ad 1 il bit 1
        OUT     2       ; Uscita risultato
; Dirama per provare la fine valida del ciclo di stampa
        JMP     LOP1
; Tabella di conteggio del ritardo
        ORG     DELY+2
        pppp            ; Ritardo di tempo H1
        qqqq            ; Ritardo di tempo H2
        rrrr            ; Ritardo di tempo H3
        ssss            ; Ritardo di tempo H4
        tttt            , Ritardo di tempo H5
        uuuu            ; Ritardo di tempo H6
```

Le lettere x, y, p, q, r, s, t ed u rappresentano valori esadecimali.

Figura 3-3. Il programma di simulazione completa.

Capitolo 4
UN SEMPLICE PROGRAMMA

I problemi associati con la simulazione della logica digitale, come è stato fatto al Capitolo 3 possono essere attribuiti al fatto seguente: si è provato a dividere la logica digitale in un certo numero di funzioni di trasferimento isolate, ognuna di queste corrisponde ad un dispositivo logico digitale. Si vuole ora abbandonare la logica digitale e combinatoria come se non esistesse ed osservare in modo diverso le Figure 3-1 e 3-2.

TIMING DEL LINGUAGGIO ASSEMBLY IN FUNZIONE DEL TIMING DELLA LOGICA DIGITALE

FUNZIONE DI TRASFERIMENTO

Ritornando alla Figura 3-1 si ignori semplicemente tutto ciò che esiste tra i margini sinistro e destro della Figura. Quello che resta è un insieme di segnali d'ingresso e di segnali d'uscita. I segnali d'uscita sono legati ai segnali d'ingresso da un insieme di funzioni di trasferimento che non hanno nulla a che fare con i dispositivi della logica digitale.

Le funzioni di trasferimento per la Figura 3-1 sono vagamente rappresentate dal diagramma di timing della Figura 3-2. Cosa significa "vagamente rappresentate"? Significa che il timing che lega alle caratteristiche del sistema è indiscriminatamente mescolato al timing che riflette semplicemente le richieste della logica digitale. Si possono abbandonare le considerazioni di timing che riflettono semplicemente a richieste della logica digitale. Specificatamente il martelletto deve essere alimentato dall'uscita di uno dei sei impulsi inviato al solenoide; i vari movimenti e ritardi di posizionamento devono essere mantenuti. Ma si possono abbandonare i ritardi di tempo che separano il cambiamento di stato di un segnale da un altro semplicemente per mantenere la logica digitale libera da rumore.

Perciò dal punto di vista del programmatore il diagramma di timing illustrato in Figura 4-1 è una sostituzione perfettamente valida del diagramma di timing del progettista logico illustrato in Figura 3-2.

SEGNALI D'INGRESSO E D'USCITA

Osservando la Figura 4-1 si vedrà che è stata abbandonata la maggior parte dei ritardi di tempo secondari; è stata anche abbandonata la maggior parte dei segnali. Ma esiste un semplice criterio per la determinazione se un segnale è effettivamente necessario all'interno di un sistema a microcalcolatore. Il criterio è questo: se il segnale è unicamente associato con gli eventi in tempo reale alla logica esterna al sistema a microcalcolatore, allora il segnale deve rimanere. Se la sorgente e la destinazione del segnale sono all'interno della "scatola nera" del sistema a microcalcolatore allora il segnale deve essere abbandonato. Basandosi su questo criterio si dia un altro sguardo ai segnali d'ingresso e uscita.

Prima si considerino i segnali d'ingresso.

SEGNALI D'INGRESSO

RETURN STROBE e PW STROBE sono segnali non significativi. Questi due segnali, per la logica digitale sono iniziatori della sequenza del ciclo di stampa. All'interno di un programma in linguaggio assembly è richiesto semplicemente il salto alla prima istruzione di una sequenza. Il fatto che RETURN STROBE rappresenti un ciclo di stampa durante il quale il martelletto non è alimentato non ha importanza perchè HAMMER ENABLE viene impiegato per sopprimere effettivamente HAMMER PULSE.

Si combineranno i vari segnali che inibiscono l'alimentazione del martelletto in un ingresso di stato del martelletto. Tali segnali sono cinque PFL REL, RIB LIFT RDY, RIBBON ADVANCE, PFR REL e CA REL. Ognuno di questi segnali deve la sua origine a logica esterna diversa alla Figura 3-1; nella realizzazione a logica digitale questi segnali subiscono l'AND in modo da creare un segnale principale HAMMER INTERLOCK. Nella realizzazione in un linguaggio assembly si cableranno OR tutti questi segnali esterni ad un singolo pin che diventa uno stato HAMMER INTERLOCK.

RESET sarà conservato come segnale RESET principale collegato al pin CPU RESET. RESET può perciò essere ignorato dal programma in linguaggio assembly; comunque si ricordi che una volta che RESET è stato attivato l'esecuzione del programma riprende con l'istruzione immagazzinata alla locazione di memoria 0.

$\overline{\text{EOR DET}}$ sarà conservato. Questo è il segnale che rivela la fine del nastro e previene ogni interruzione del ciclo di stampa, inibendo in questo modo la stampa di un ulteriore carattere dopo che il nastro è terminato.

$\overline{\text{HAMMER ENABLE FF}}$ deve essere conservato; esso sopprime l'impulso di alimentazione del martelletto durante il ciclo di riposizionamento della ruota.

La funzione eseguita dai sei segnali di lunghezza dell'impulso del martelletto da H1 ad H6 deve rimanere ma gli stessi segnali scompariranno. Invece di impiegare sei pin di una Porta I/O per identificare la larghezza dell'impulso del martelletto si creano direttamente dei ritardi di tempo dai codici dei caratteri ASCII.

Si ponga ora l'attenzione ai segnali d'uscita.

Si possono eliminare tutte le uscite dei flip-flop. Il limite di ogni intervallo di tempo all'interno di un ciclo di stampa è già identificato dal cambio di stato di un segnale esistente. Se si deve far scattare più di un evento della logica esterna mediante una transizione da un intervallo di tempo al successivo non c'è che da arrestare il segnale opportuno su un buffer esterno e quindi impiegarlo per far scattare numerosi eventi logici esterni. All'interno di un programma al microcalcolatore non ci sono ragioni per duplicare segnali che dovrebbero semplicemente identificare la transizione da un intervallo di tempo del ciclo di stampa al successivo.

I rimanenti segnali d'uscita sono conservati. E' possibile che alcuni di questi segnali scompaiano se la logica addizionale esterna fosse sostituita da più programmi in linguaggio assembly all'interno del sistema a microcalcolatore; ma, dati i limiti del problema, come stabilito, i segnali rimanenti sono necessari allo scopo di definire gli intervalli di tempo del ciclo di stampa.

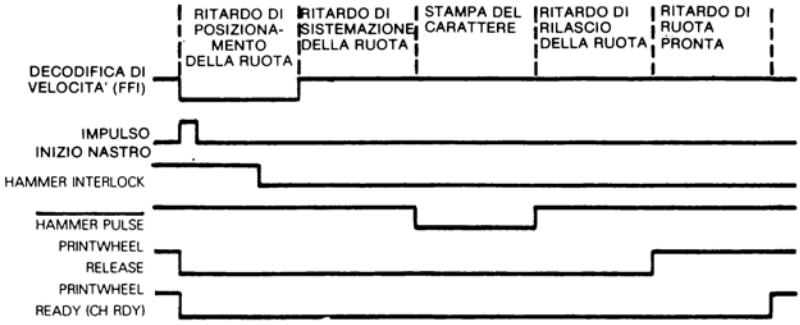
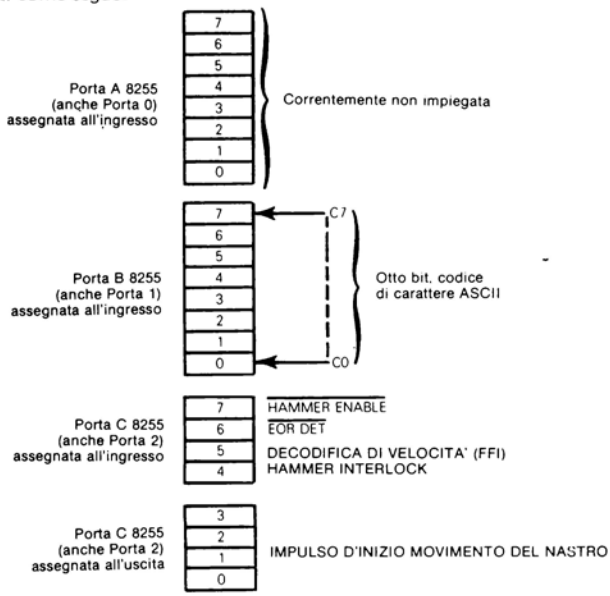


Figura 4-1. Timing per la Figura 3-1 dal punto di vista del programmatore

ASSEGNAZIONE DI PIN

Dato il nuovo set di segnali, semplificato, si può eliminare il buffer I/O 8212 ed impiegare un 8255 Interfaccia Periferica Programmabile (PPI), funzionante nel Modo 0, con le porte I/O ed i pin assegnati come segue:



CONFIGURAZIONE DEL DISPOSITIVO A MICROCALCOLATORE

E' ora possibile selezionare i dispositivi necessari per la realizzazione logica. La selezione in realtà è abbastanza diretta; oltre alla CPU sarà necessaria un'Interfaccia Periferica Programmabile (PPI) 8255, qualche memoria di sola lettura per l'immagazzinamento del programma e qualche memoria di lettura-scrittura per l'immagazzinamento generale dei dati. La CPU, in realtà consiste di 3 dispositivi, la CPU stessa il chip Clock 8224 ed il Bus Controller 8228. Combinando questi dispositivi la Figura 4-2 illustra il sistema a microcalcolatore risultante. Ora se non si comprende immediatamente la Figura 4-2 non ci si preoccupi infatti ci sono solo alcuni aspetti di questa figura che sono conseguenti alla discussione immediata.

CONCETTI DI PROGETTO GENERALE

Questo è il concetto più importante da derivare dalla Figura 4-2: quando si considera la progettazione mediante scrittura di programmi in linguaggio assembly all'interno di un sistema a microcalcolatore, il programma scritto dipende fortemente dalla configurazione del dispositivo. Il modo di combinare i dispositivi illustrati in Figura 4-2 non è unico, configurazioni alternative potrebbero essere ugualmente percorribili. I programmi in linguaggio assembly, possono comunque creare notevoli differenze tra una configurazione del microcalcolatore e la successiva e questo è un fattore che non si dovrebbe perdere di vista nella scrittura di programmi al microcalcolatore. Inoltre non ci si preoccupi della modificazione della configurazione hardware selezionata; questo è precisamente quanto si farà al

Capitolo 5. **La configurazione del dispositivo a microcalcolatore e la programmazione in linguaggio assembly interagiscono fortemente e non dovrebbero essere separati.** Queste due fasi dovrebbero essere all'interno dell'anello iterativo. Durante i primi stadi scrittura di un programma al microcalcolatore si dovrebbe assumere che nel corso della scrittura del programma in linguaggio assembly si scopriranno le caratteristiche dell'hardware che possono essere migliorate, che quindi implicano la riscrittura del programma.

LINGUAGGI AD ALTO LIVELLO

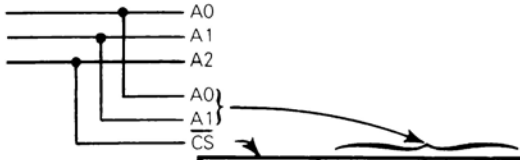
Questa è una delle ragioni principali per cui i programmi ad alto livello non sono desiderabili quando si sta programmando la sostituzione della logica digitale con un microcalcolatore. I linguaggi al livello più elevato sono orientati al problema. Per esempio è difficile, osservando uno statement del programma PL/M, visualizzazione il modo esatto in cui i dati saranno mossi nel sistema a microcalcolatore in risposta all'esecuzione dello statement. Inoltre è difficile collegare i programmi PL/M alle configurazioni esatte dei dispositivi. Il linguaggio assembly, d'altra parte, ha una relazione uno-ad-uno con l'hardware.

INTERFACCIA PERIFERICA PROGRAMMABILE (PPI) 8255

Ora si ponga l'attenzione al modo specifico in cui i dispositivi sono stati inseriti nella Figura 4-2.

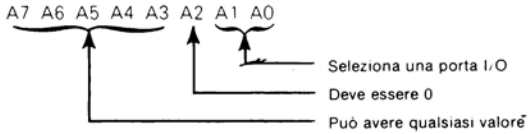
SELEZIONE DELLA PORTA I/O

L'Interfaccia Periferica Programmabile 8255 risponderà agli indirizzi della Porta I/O 0, 1 e 2 per le Porte A, B e C, rispettivamente. Questo perchè il chip selezionato è collegato alla linea A2 del bus indirizzo. Poichè l'indirizzo della porta I/O esce sulle otto linee di basso ordine del bus indirizzo quando viene eseguita un'istruzione IN o OUT, l'8255 PPI in Figura 4-2 risponderà agli indirizzi della porta I/O come segue:



$\overline{CS}$	A1	A0	Porta I/O selezionata
0	0	0	Porta A
0	0	1	Porta B
0	1	0	Porta C (superiore ed inferiore)
0	1	1	Porta di Controllo

$\overline{CS}$ deve essere 0 perchè sia selezionata la PPI. Questo significa che, come mostrato in Figura 4-2, la PPI sarà selezionata ogni volta che A2 è 0, non considerando come possano essere le linee da A3 ad A7. **Così la PPI risponderà a qualsiasi indirizzo della porta I/O escludendo 4, 5, 6 e 7.**



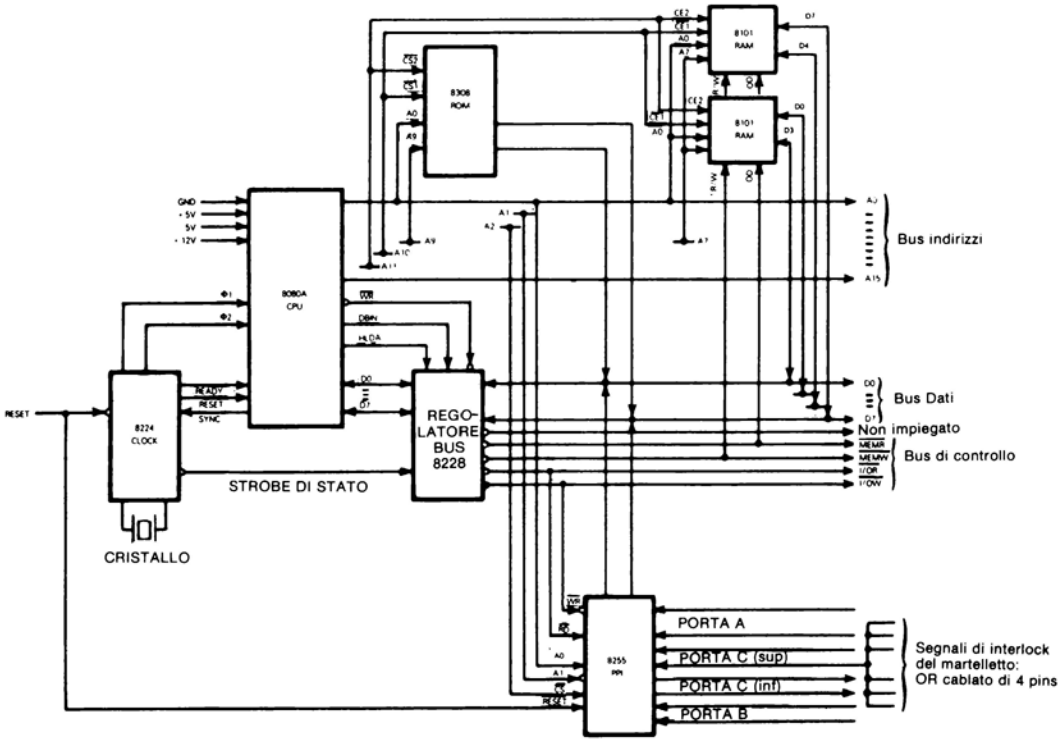


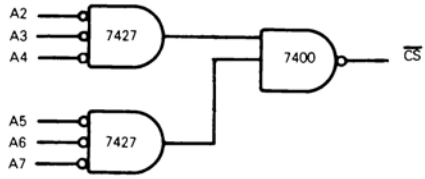
Figura 4-2. Configurazione del microcalcolatore

SELEZIONE DEL CHIP IN SISTEMI SEMPLICI

Se la configurazione di un microcalcolatore contiene un grande numero di Interfacce Periferiche Programmabili (PPI) la logica del chip selezionato deve diventare un po' più complessa. Se una PPI deve rispondere per quattro indirizzi di porta I/O, escludendo tutti gli altri, allora l'ingresso del chip selezionato deve essere originato combinando le sei linee di indirizzo di ordine elevato in un modo unico. Naturalmente questo implica che il sistema a microcalcolatore contenga 64 PPI — e questo è improbabile.

SELEZIONE DEL CHIP IN SISTEMI PIU' GRANDI

Si supponga che l'8255 PPI della Figura 4-2 debba rispondere solo agli indirizzi delle Porte I/O 0, 1, 2 e 3. Ecco un modo di originare l'ingresso di selezione del Chip ($\overline{CS}$):



Se e solo se tutte le sei linee d'indirizzo da A2 ad A7 sono basse, allora $\overline{CS}$ sarà basso e l'8255 PPI sarà selezionato.

Ora la direzione dei dati e l'utilizzazione della porta illustrata in Figura 4-2 per l'8255 PPI non è una caratteristica hardware. In qualsiasi istante l'utilizzazione della porta può essere modificata dalla scrittura di un'appropriata parola di controllo nella Porta I/O 3 che è la porta di controllo per l'8255 PPI così configurato.

LOGICA DI RESET

Anche la logica di RESET richiede commento. Invece di verificare una condizione di reset tra cicli di stampa, come è stato fatto al Capitolo 3, **si impiegherà un segnale di RESET hardware** ma in un sistema a microcalcolatore. Il segnale RESET, essendo connesso al clock 8224 ed all'8255 PPI azzererà tutti i registri dell'8255 PPI ed il Contatore di Programma della CPU 8080. Questo significa che l'esecuzione del programma riprenderà con l'istruzione immagazzinata nel byte di memoria il cui indirizzo è 0. **Si devono perciò incominciare le fasi di post-reset ed inizializzazione del sistema da questa locazione di memoria.**

INIZIALIZZAZIONE DEL SISTEMA

Quando il sistema è inizializzato, la condizione "tra cicli di stampa" deve essere immediatamente ristabilita. Le fasi necessarie sono queste:

1) Se il martelletto è stato alimentato, si deve interrompere l'impulso di alimentazione o consentire il tempo al martelletto di ritirarsi.
2) Rimuovere la ruota di stampa alla sua posizione di visibilità.
3) Assicurarsi che i segnali d'uscita si trovino nello stato "tra cicli di stampa".

SEQUENZA DI REALIZZAZIONE DEL PROGRAMMA

Si è così prevenuti ad un altro concetto fondamentale della programmazione: **esiste una sequenza "più efficiente" secondo la quale si dovrebbero scrivere i programmi sorgenti in linguaggio assembly.** Si potrebbe procedere scrivendo un programma di inizializzazione per realizzare un RESET ma questo richiederebbe molte ipotesi. Come conoscere se il martelletto è stato alimentato? Come rimuovere la ruota di stampa alla sua posizione di visibilità? RESET interromperà un ciclo di stampa - perciò il programma del ciclo di stampa deve essere originato prima di conoscere come interromperlo.

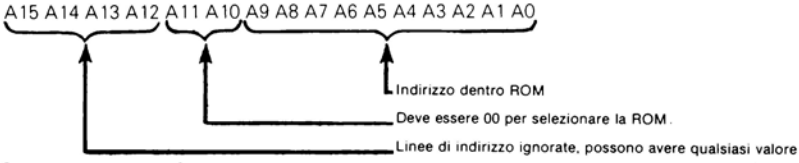
Generalmente è stabilito che si potrebbe iniziare la scrittura di un programma realizzando gli eventi più importanti della logica e quindi, sulla base di questo lavoro, realizzare gli eventi dipendenti.

In particolare ora si considererà prima la realizzazione della logica di RESET e poi la realizzazione di un programma finché non è stato originato il programma del ciclo di stampa.

MEMORIA ROM E RAM

ROM

L'impiego di un segnale RESET significa che deve esserci un byte di memoria con indirizzo 0. Nella Figura 4-2 questo byte di memoria farà parte della ROM 8308. Questa è una ROM di 1024 byte; essa corrisponderà agli indirizzi di memoria da 0 a $03FF_{16}$ poichè le sue linee di selezione del chip sono collegate alle linee del bus indirizzo A10 ed A11. Questo può essere illustrato come segue:



INDIRIZZI DI MEMORIA

SELEZIONE DI ROM IN SISTEMI SEMPLICI

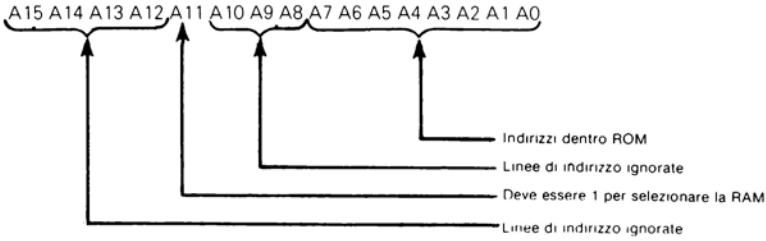
Ancora una volta si sta impiegando la selezione di un chip ROM primitivo considerata la semplicità del sistema a microcalcolatore. Si definisce il campo di indirizzo da 0 a $03FF_{16}$ per i 1024 bytes della memoria ROM, infatti una grande quantità di altri indirizzi potrebbe anche accedere alla memoria ROM - le linee di indirizzo da A12 ad A15 potrebbero avere qualunque valore. Fornendo A10 ed A11 entrambe a 0 si può accedere alla memoria ROM. Non c'è nulla per prevenire la selezione della memoria nel modo primitivo, fornito in un piccolo sistema a microcalcolatore. Non c'è ragione per incorrere in spese addizionali creando codici complessi di selezione dei chip impieganti tutte le linee di ordine elevato del bus indirizzo. Anche dal punto di vista della programmazione non si **dovrà riscrivere il programma espandendo il sistema e comprendendo in seguito più memoria. Purchè non si impieghi ora nessuno degli indirizzi alternativi che la ROM può selezionare allora in futuro si potrà prendere uno di questi set alternativi di indirizzi, impiegarlo per selezionare una altra ROM e non influenzerà in alcun modo i programmi già scritti.**

Specificando la ROM per l'immagazzinamento dei dati si assumerà che il prodotto sia sufficientemente sviluppato in volume da giustificare il dispendio della creazione di una maschera ROM. Se il volume non giustifica questo dispendio allora si può impiegare la Memoria a Sola Lettura Programmabile (PROM).

RAM

INDIRIZZI DI MEMORIA

I due dispositivi RAM 8101 forniscono 1024 bit ciascuno di memoria di lettura/scrittura, organizzati in 256 unità a 4 bit. Ogni RAM perciò fornisce metà di un byte di memoria di lettura/scrittura. **I 256 byte RAM avranno indirizzo da 0800_{16} a $08FF_{16}$. Questo può essere illustrato come segue:**



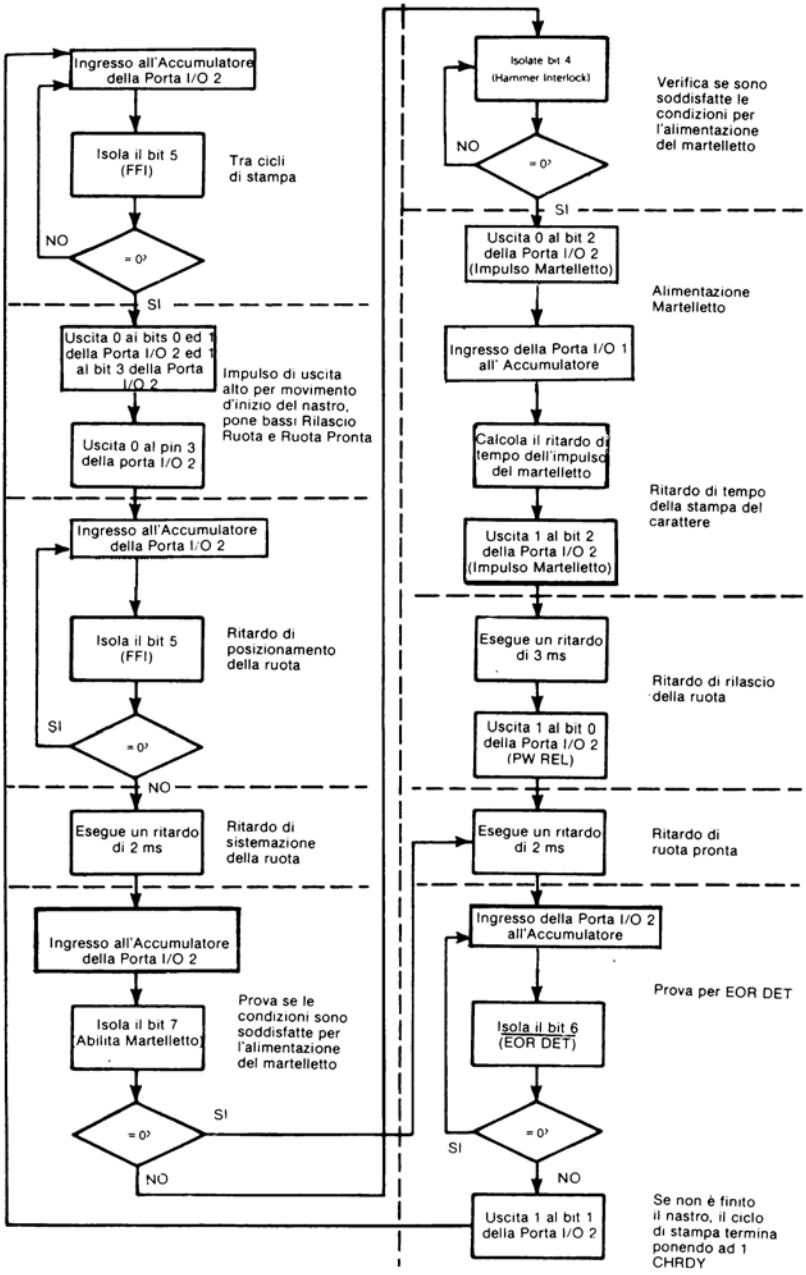


Figura 4-3. Primo tentativo di costruzione del diagramma di flusso del programma.

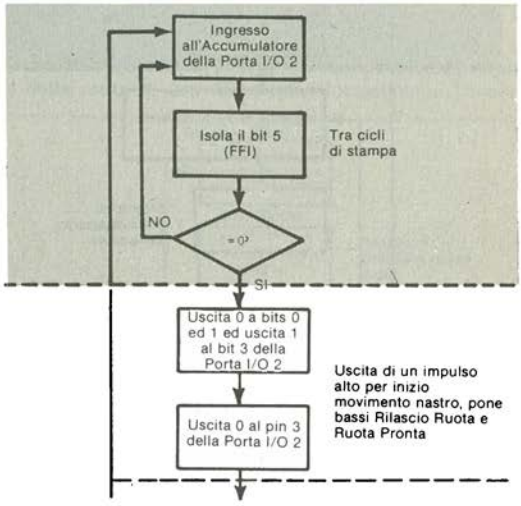
Anche se gli indirizzi di memoria da 0800_{16} a $08FF_{16}$ sono stati specificati fornendo lo spazio indirizzo RAM, ancora una volta una grande quantità di altri indirizzi potrebbe anche selezionare RAM. Si noti comunque che, in nessun caso, un indirizzo RAM coinciderà con un indirizzo ROM; la linea del bus indirizzo A11 deve essere sempre 0 per selezionare ROM mentre deve essere sempre ad 1 per selezionare RAM. Quindi non sorgeranno mai conflitti di indirizzo.

Altre caratteristiche della Figura 4-2 non sono significative alla generazione del programma al livello attualmente in discussione, perciò non è necessaria la comprensione della configurazione hardware in qualsiasi dettaglio ulteriore.

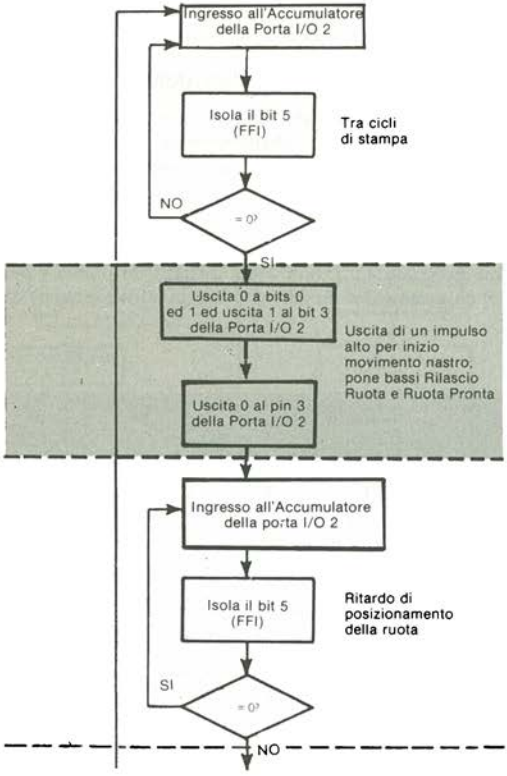
DIAGRAMMA DI FLUSSO DEL PROGRAMMA

Si ponga ora l'attenzione sulle funzioni che devono essere eseguite dal sistema a microcalcolatore. Queste funzioni sono identificate dal diagramma di flusso illustrato in Figura 4-3. Si analizzerà, fase-per-fase, questo diagramma di flusso.

Si impiegherà il segnale d'ingresso di decodifica della velocità (FFI) per identificare l'inizio di un nuovo ciclo di stampa. Perciò tra cicli di stampa il programma fa entrare continuamente all'Accumulatore i contenuti della Porta I/O 2, verificando il bit 5. Finchè questo bit è uguale ad 1 non viene iniziato un nuovo ciclo di stampa. Non appena questo bit è uguale a 0 viene identificato un nuovo ciclo di stampa:

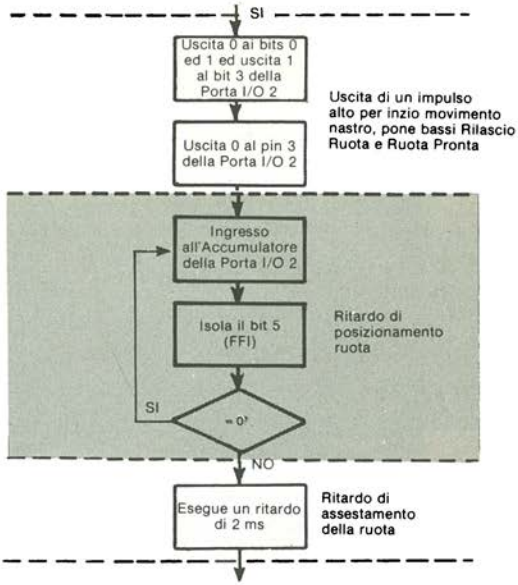


La prima cosa che accade all'interno di un nuovo ciclo di stampa è che esce **un impulso alto di INIZIO MOVIMENTO DEL NASTRO dalla scrittura sequenziale di un 1, poi di uno 0 al bit 3 della Porta I/O 2. Inoltre gli zeri escono ai bit 0 ed 1 della Porta I/O 2,** poichè PRINTWHEEL RELEASE e PRINTWHEEL READY devono entrambi uscire bassi all'inizio del ciclo di stampa:

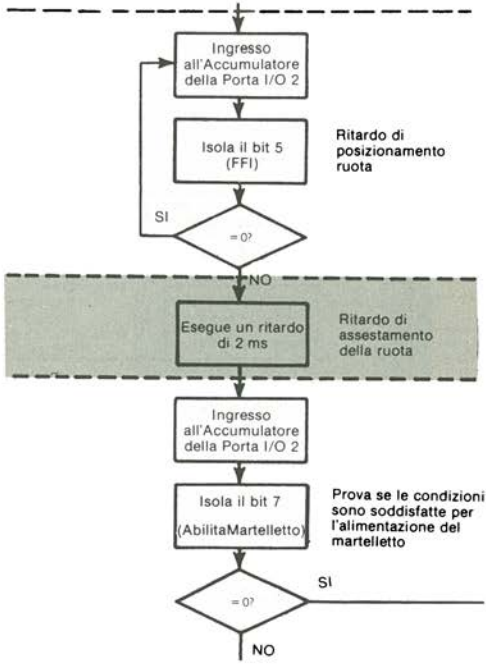


Il ritardo di riposizionamento della ruota di stampa è calcolato dal segnale di decodifica della velocità FFI. Mentre questo segnale è basso la ruota di stampa deve essere ancora posizionata. Quindi si entra in un anello di ritardo variabile che, in termini del programma, è l'inverso dell'anello di ritardo "tra cicli di stampa". Ancora una volta i contenuti della Porta I/O 2 entrano nell'Accumulatore ed è verificato il bit 5; comunque si permane nell'anello di ritardo finchè il bit 5 è 1. In quell'istante termina il ritardo

di posizionamento della ruota di stampa:

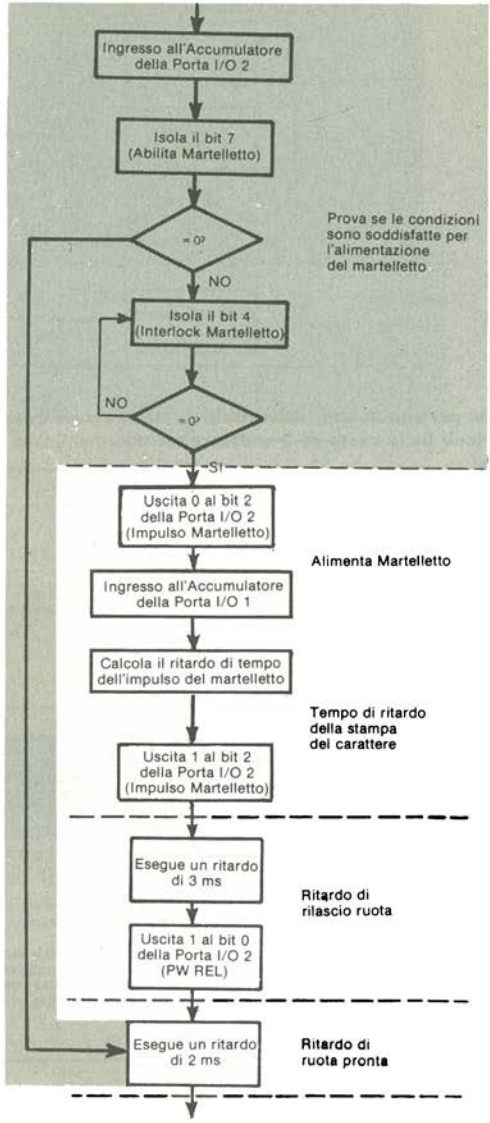


Il ritardo del posizionamento della ruota di stampa deve essere seguito da un ritardo di sistemazione della ruota di 2 millisecondi. Normalmente l'anello di ritarso sarà eseguito così:



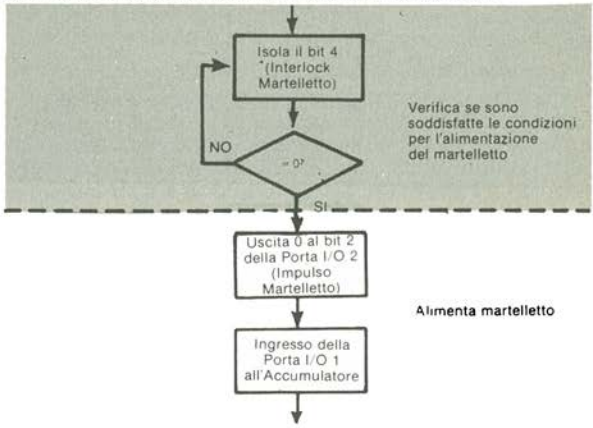
Alla fine del ritardo di sistemazione della ruota, **viene alimentato il martelletto, rendendo basso il segnale HAMMER INTERLOCK ed alto HAMMER ENABLE.** Si ricordi che HAMMER INTERLOCK è un singolo bit di stato, impiegato da tutte le condizioni esterne che possono prevenire l'alimentazione del martelletto. Qualunque segnale inviante un livello alto a questo pin di stato sopprimerà l'alimentazione del martelletto.

Un ciclo di riposizionamento del martelletto è identificato dall'ingresso basso di HAMMER ENABLE. Questa condizione è rivelata isolando il bit 7 della Porta I/O 2

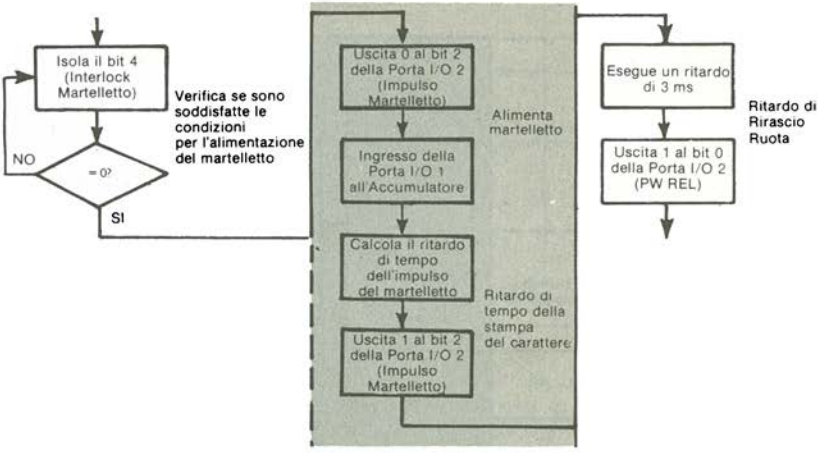


prima di verificare la condizione di HAMMER INTERLOCK. Se il bit 7 della Porta I/O 2 è 0 allora l'intera sequenza di alimentazione del martelletto viene scavalcata e si salta direttamente al ritardo di ruota pronta, che è l'ultimo ritardo di tempo del ciclo di stampa come disegno di pag. 4-12.

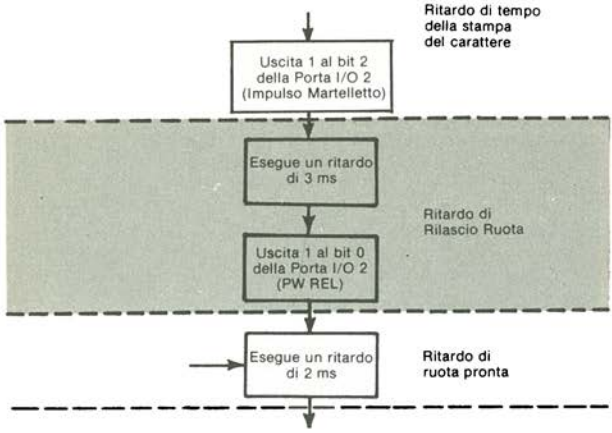
Se HAMMER ENABLE è alto si ha il ciclo di stampa di un carattere cosicchè il martelletto sarà alimentato ma solo quando HAMMER INTERLOCK è zero. Mentre qualunque segnale calcolato OR al pin 4 della Porta I/O 2 è alto il programma rimarrà in un anello senza fine verificando continuamente lo stato di questo pin della porta I/O. Quando finalmente il pin della porta I/O è uguale a 0 il programma procederà alla sequenza d'istruzione di alimentazione del martelletto:



Allo scopo di alimentare il martelletto si deve far uscire un impulso di alimentazione di lunghezza variabile. Per fare questo viene fatto uscire uno 0 dal pin 2 della Porta I/O 2, poichè questo è il pin attraverso il quale esce l'impulso del martelletto. Si descriverà come viene calcolata la lunghezza dell'impulso del martelletto una volta completata la descrizione del diagramma di flusso.
Alla fine del ritardo di tempo di alimentazione del martelletto viene fatto uscire un 1 dal bit 2 della Porta I/O 2. Questo termina l'impulso di alimentazione del martelletto:

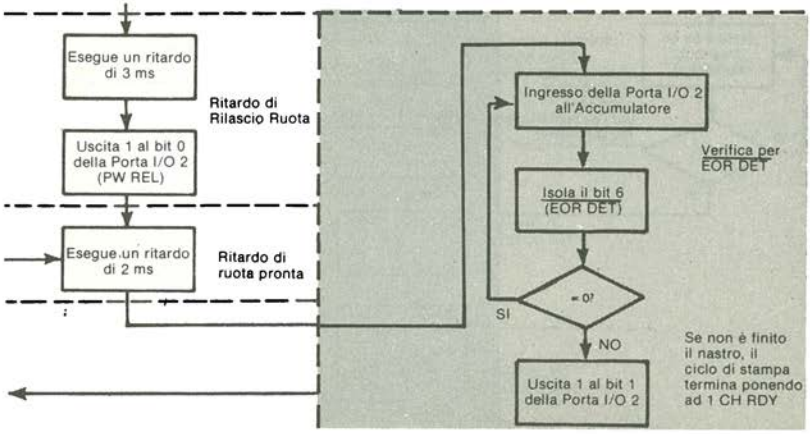


Ora seguono ritardi di assestamento. Prima c'è un ritardo di 3 millisecondi per liberare la ruota di stampa, il cui termine è contrassegnato da un 1 che esce dal bit 0 della Porta I/O 2. Questo fa uscire alto PW REL:



Successivamente viene eseguito un ritardo di 2 millisecondi di ruota pronta. La fine di questo ritardo **e la fine del ciclo di stampa sono contrassegnate dall'uscita di un 1 al bit 1 della Porta I/O 2;** questo pone alto CH RDY. **Comunque non si vuole fare questo se c'è uno stato di termine del nastro.** Questo stato è identificato dal divenire basso di $\overline{\text{EOR DET}}$.

Il programma perciò fa entrare la Porta I/O 2 ed isola il bit 6 attraverso il quale $\overline{\text{EOR DET}}$ è fatto entrare nel sistema a microcalcolatore. Se $\overline{\text{EOR DET}}$ è uguale a 0 allora il programma permane in un ciclo senza fine riverificando continuamente il bit 6 della Porta I/O 2, cosicchè può iniziare un altro ciclo di stampa. Solo se $\overline{\text{EOR DET}}$ è rivelato uguale a 1 terminerà il ciclo di stampa con la posizione ad 1 di CH RDY:

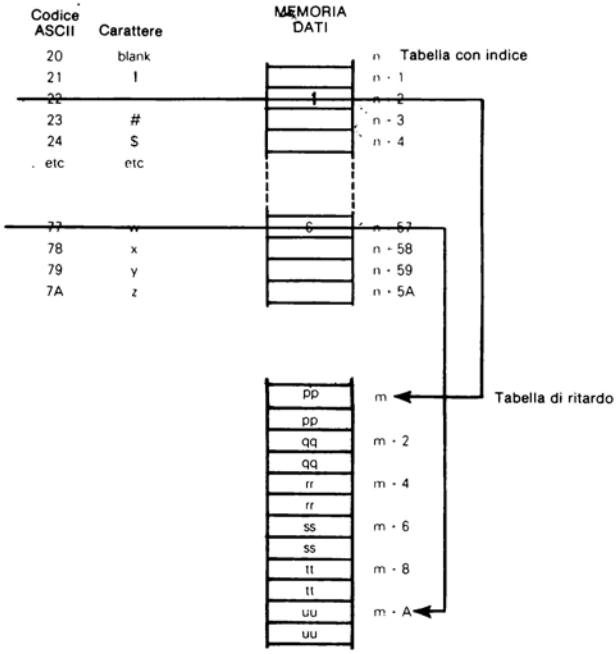


RITARDO DI ALIMENTAZIONE DEL MARTELLETTO

Si ponga ora l'attenzione al metodo attraverso il quale viene calcolato il ritardo opportuno di alimentazione del martelletto. Nella Figura 3-1 il ritardo opportuno di alimentazione del martelletto è stato segnalato da una delle sei linee (da H1 ad H6) che entra al livello alto. Questa linea viene generata al livello alto dalla logica esterna in relazione alla natura del carattere da stampare; **questo tipo di operazione è facile da realizzare all'interno di un programma al microcalcolatore.**

Questo è il metodo che sarà impiegato per calcolare l'appropriato ritardo di tempo dell'impulso di alimentazione del martelletto: ogni carattere da stampare è rappresentato da un byte dati del codice ASCII, come illustrato nell'Appendice A.

Se si ignora il bit parità di ordine elevato allora rimangono 128 combinazioni di bit possibili. Se si osservano i codici ASCII forniti nell'Appendice A si vedrà che solo i codici di carattere tra 20_{16} e $7A_{16}$ sono significativi. Perciò solo le combinazioni di codice $5A_{16}$ (o 90_{10}) devono essere prese in considerazione. Ognuna di queste combinazioni sarà assegnata ad un byte in una tabella di 90-byte; inoltre in questo byte sarà immagazzinato un numero tra 1 e 6. Questo numero identificherà il ritardo di tempo richiesto dal carattere. Una tabella a 12-byte conterrà i sei ritardi di tempo effettivi associati con i sei digit. Questo schema può essere illustrato come segue:



Nella precedente illustrazione le lettere "n" ed "m" alla destra della memoria dati rappresentano qualsiasi indirizzo valido della memoria di base. Per esempio, "n" può rappresentare 0380_{16} mentre "m" rappresenta $03F0_{16}$.

Si considerino due esempi.

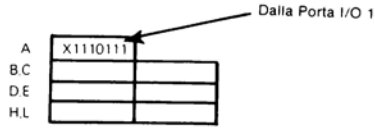
Il codice ASCII 22_{16} indica il carattere di doppia virgoletta (") che richiede il ritardo di tempo più breve. Il byte della memoria dati con indirizzo n+2 corrisponde a questo codice ASCII. In questo byte della memoria dati è immagazzinato 1. Perciò il primo ritardo di tempo, rappresentato da pppp, è il valore ritardo di tempo breve che origina l'impulso di alimentazione del martelletto per il carattere".

Il codice ASCII 77_{16} rappresenta "w". Il byte della memoria dati con indirizzo n+57_{16} corrisponde a questo codice ASCII. All'interno di questo byte della memoria dati viene immagazzinato il valore 6, che significa che "w" richiede il ritardo di alimentazione del martelletto più lungo. Perciò un valore rappresentato da uuuu sarà caricato nei registri D ed E prima dell'esecuzione dell'anello del ritardo di tempo lungo che origina l'impulso di alimentazione del martelletto per il carattere w.

La Figura 4-4 identifica la fase del programma attraverso il quale il ritardo di alimentazione del martelletto verrà calcolato.

Allo scopo di capire meglio la Figura 4-4 si andrà dalla fase Ⓐ alla Ⓘ per il caso "w".

Ⓐ La rappresentazione ASCII del caso più basso w viene inviata all'Accumulatore:



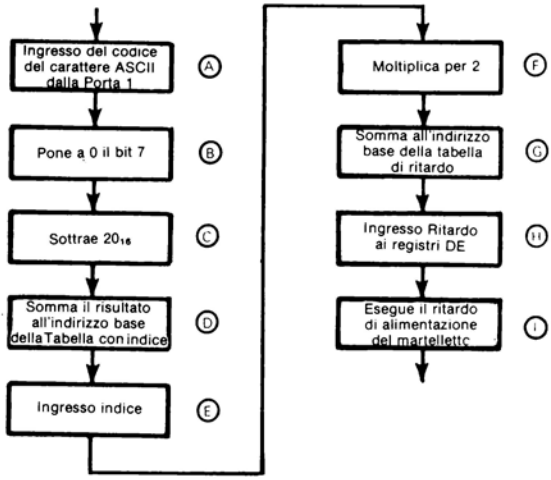
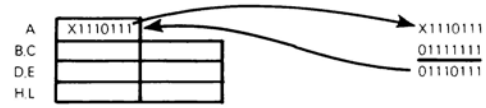


Figura 4-4. Diagramma di flusso del programma per calcolare la lunghezza dell'impulso di alimentazione del martelletto

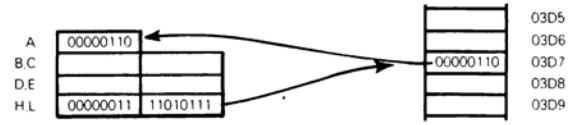
Ⓑ Si deve porre il bit parità a 0. Per fare questo si opera l'AND dei contenuti dell'Accumulatore con 7FH:



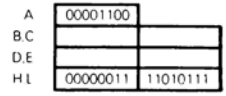
Ⓒ L'ingresso della tabella con indice corrispondente al caso più basso w è calcolato aggiungendo il codice ASCII meno 20_{16} all'indirizzo base della tabella con indice. Si deve sottrarre 20_{16} perchè il primo codice 1F non ha un equivalente ASCII:

Ⓓ L'indirizzo base della tabella con indice è caricato nei registri H ed L. Si assumerà che questo indirizzo sia 0380_{16}. Allora i contenuti dell'Accumulatore sono sommati a questo indirizzo a 16-bit:

Ⓔ L'indice appropriato è caricato dalla tabella con indice nell'Accumulatore:

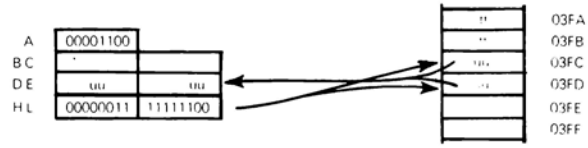


Ⓕ Poichè il ritardo effettivo è lungo due byte, si calcola l'indirizzo del ritardo appropriato aggiungendo il doppio indice all'indirizzo base della tabella ritardo. Prima si deve moltiplicare l'indice per 2:



Ⓖ Successivamente si somma l'indice moltiplicato per 2 all'indirizzo base della tabella di ritardo. Si assume che questo indirizzo base è $03F0_{16}$. Questo indirizzo base viene ancora caricato nei registri H ed L, dopo che i contenuti dell'Accumulatore sono stati sommati ai contenuti dei registri H ed L:

Ⓗ I due bytes indicizzati da H ed L sono caricati nei registri D ed E:



Ⓘ I registri D ed E ora contengono il valore iniziale corretto per l'esecuzione di un ritardo lungo come descritto al Capitolo 2.

```
; Programma ciclo di stampa
; Prova FFI tra cicli di stampa (per il valore 0 del bit 5 della Porta I/O 2)
START IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI     20H     ; Isola il bit 5
      JNZ     START   ; Se non 0 ritorna a START
; Inizializza il ciclo di stampa, uscita 0 ai bit 0
; Ed 1 della Porta I/O 2, uscita 1 al bit 3 della Porta I/O 2
      MVI     A,0CH   ; Carica la maschera nell'Accumulatore
      OUT     2       ; Uscita alla Porta I/O 2
; Uscita 0 al bit 3 della Porta I/O 2. Questo completa
; L'impulso d'INIZIO MOVIMENTO DEL NASTRO
      MVI     A,4     ; Carica la maschera nell'Accumulatore
      OUT     2       ; Uscita alla Porta I/O 2
; Prova per termine posizionamento ruota di stampa
LOP1  IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      ANI     20H     ; Isola il bit 5
      JZ      LOP1    ; Se 0 ritorna a LOP1
; Esegue il ritardo di 2ms per la sistemazione della ruota
      MVI     A,0     ; Carica l'Accumulatore con 0
LOP2  DCR     A       ; Decrementa A
      JNZ     LOP2    ; Se A non è decrementato a 0 ri-decrementa
; Verifica delle condizioni di alimentazione del martelletto
LOP3  IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
      RLC             ; Muove il bit 7 in Carry
      JNC     PRD     ; Se Carry è 0 scavalca l'alimentazione del martelletto
      ANI     20H     ; Isola il bit 4 che ora è il bit 5
      JZ      LOP3    ; Attende per valore non zero prima dell'alimentazione
; Alimenta martelletto
      IN      2       ; Pone basso l'impulso del martelletto uscita 0
      ANI     FBH     ; Al bit 2 della Porta I/O 2
      OUT     2
      IN      1       ; Ingresso del carattere ASCII all'Accumulatore
      ANI     7FH     ; Maschera il bit di ordine elevato
      SUI     20H     ; Sottrae 20H
      LXI     H,INDX  ; Carica ad HL l'indirizzo base della tabella con indice
      ADD     L       ; Somma ad HL i contenuti dell'Accumulatore
      MOV     L,A
      MOV     A,M     ; Carica l'indice nell'Accumulatore
      ADD     A       ; Moltiplica per 2
      LXI     H,DELY  ; Carica l'indirizzo base della tabella di ritardo in HL
      ADD     L       ; Somma ad HL i contenuti dell'Accumulatore
      MOV     L,A
      MOV     E,M     ; Carica la costante di ritardo in DE
      INX     H
      MOV     D,M
LOP4  DCX     D       ; Esegue il ritardo lungo
      MOV     A,D
      ORA     E
      JNZ     LOP4
```

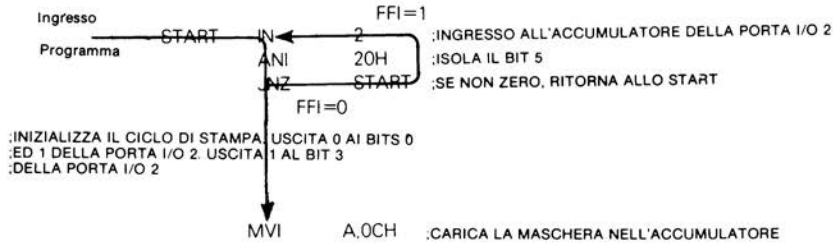
Figura 4-5. Una semplice sequenza d'istruzione del ciclo senza inizializzazione o reset

```
        IN      2       ; Alla fine del ritardo fa uscire 1 al bit 2
        ORI     4       ; Della Porta I/O 2 (impulso del martelletto alto)
        OUT     2
; Esegue un ritardo di tempo di rilascio ruota di 3ms
        LXI     D,F7H   ; Carica la costante di tempo in D,E
LOP5    DCR     D       ; Decrementa D, E
        MOV     A,D     ; Verifica se c'è 0 in D,E
        ORA     E
        JNZ     LOP5    ; Ridecrementa se non zero
; Uscita 1 al bit 0 della Porta I/O 2. Questo pone alto PW REL
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     1       ; Pone ad 1 il bit 0
        OUT     2       ; Uscita risultato
; Esegue un ritardo di ruota pronta di 2 millisecondi
PRD     MOV     A,0     ; Carica l'Accumulatore con 0
LOP6    DCR     A       ; Decrementa A
        JNZ     LOP6    ; Se A non è decrementato a 0, lo ri-decrementa
; Prova per EOR DET (bit 6 della Porta I/O 2) uguale
; A 0 come un pre-requisito per terminare il ciclo di stampa
LOP7    IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ANI     40H     ; Isola il bit 6
        JZ      LOP7    ; Ritorno e riverifica se 0
; Alla fine del ciclo di stampa pone ad 1 il bit 1 della Porta I/O 2
. Questo pone alto CH RDY
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     2       ; Pone ad 1 il bit 1
        OUT     2       ; Uscita risultato
        JMP     START   ; Salta alla verifica del nuovo ciclo di stampa
```

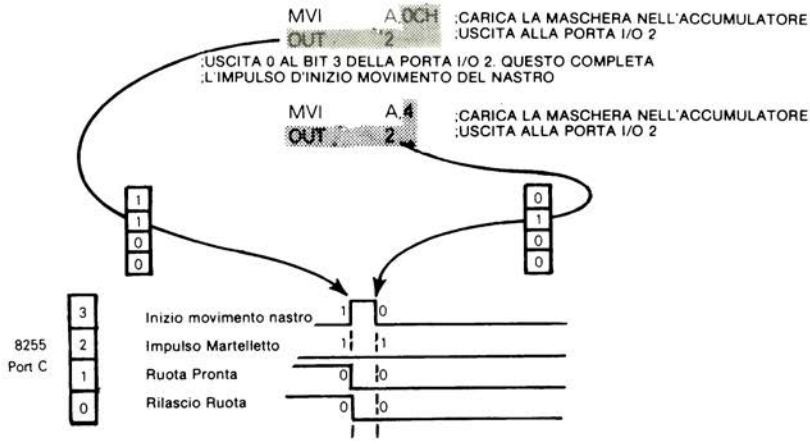
Figura 4-5. Una semplice sequenza d'istruzione del ciclo senza inizializzazione o reset (segue).

Mettendo assieme i diagrammi di flusso del programma, illustrati in Figura 4-3 e 4-4 si genera l'intero programma richiesto, come illustrato in Figura 4-5. Parte-per-parte viene ora descritto questo programma.

L'anello seguente di tre istruzioni, tra cicli di stampa, verifica continuamente lo stato del bit 5 della Porta I/O 2. Il segnale FFI è fatto entrare a questo pin. Finchè questo segnale entra alto non può iniziare un nuovo ciclo di stampa. Non appena questo segnale entra basso la ruota di stampa è ritenuta in movimento — questo significa che è in corso un nuovo ciclo di stampa:



Non appena inizia un nuovo ciclo di stampa i segnali PRINTWHEEL RELEASE e PRINTWHEEL READY devono uscire bassi. Inoltre deve uscire al livello alto l'impulso di inizio movimento del nastro cosicchè, quando viene alimentato il martelletto, di fronte al carattere da stampare viene a trovarsi del nastro fresco. Questi cambiamenti di segnale iniziali possono essere illustrati come segue:



IMPULSO DI SEGNALE PROGRAMMATO

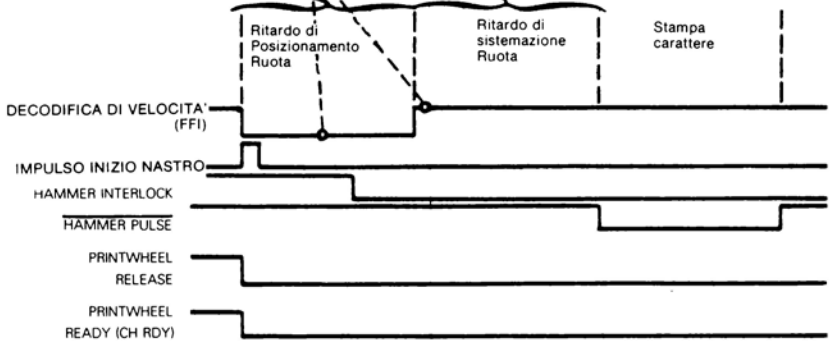
Nella precedente illustrazione si noti che il pin 2 della Porta I/O C è stato forzato ad uscire ad 1. Questo è il pin HAMMER PULSE che diventa basso solo per la durata dell'impulso di alimentazione del martelletto. A questo punto del ciclo di stampa, questo segnale è alto, così l'uscita di 1 è inefficiente.

RITARDO DI TEMPO DI LUNGHEZZA VARIABILE

Il programma ora esegue un ritardo di tempo di lunghezza variabile, durante questo tempo la ruota si muove finchè il petalo del carattere appropriato è di fronte al martelletto oppure la ruota di stampa viene rimossa alla sua

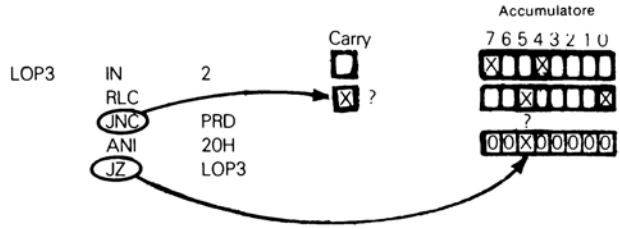
posizione di visibilità. In entrambi i casi la logica esterna fa entrare basso il segnale FFI per la durata del ritardo di posizionamento della ruota. Non appena la ruota di stampa è stata posizionata, FFI è rivelato alto — **e la logica del programma procede al ritardo di sistemazione della ruota di 2 millisecondi.** Precedentemente è stato visto questo anello di ritardo a 3 istruzioni:

```
;VERIFICA PER LA FINE DEL POSIZIONAMENTO DELLA RUOTA
LOP1    IN      2       ;INGRESSO ALL'ACCUMULATORE DELLA PORTA I/O 2
        ANI     20H     ;ISOLA IL BIT 5
        JZ      LOP1    ;SE 0 RITORNA A LOP 1
;ESEGUE IL RITARDO DI 2 MS DI SISTEMAZIONE DELLA RUOTA
        MVI     A,0     ;CARICA CON 0 L'ACCUMULATORE
LOP2    DCR     A       ;DECREMENTA A
        JNZ     LOP2    ;SE NON È DECREMENTATO A 0 LO RIDECREMENTA
;VERIFICA DELLE CONDIZIONI DI ALIMENTAZIONE DEL MARTELLETTO
```



Ora il martelletto è pronto per essere alimentato. Prima si verifica la condizione su HAMMER PULSE che è stato connesso al pin 7 della Porta i/O 2. Se questo segnale è basso allora ci si trova in un ciclo di riposizionamento della ruota e l'intera sequenza d'istruzione di alimentazione del martelletto viene scavalcata. **Se HAMMER ENABLE è alto si attraversa questa prova. Ma HAMMER INTERLOCK deve ancora essere verificato;** questo segnale entra al pin 4 della Porta I/O 2.

L'istruzione di Scorrimento muove il bit 4 (che rappresenta HAMMER INTERLOCK) al bit 5 ed il bit 7 (che rappresenta HAMMER ENABLE) nello stato Carry:



Avendo caricato nell'Accumulatore i contenuti della Porta I/O 2 una volta, è stata verificata in modo seriale la condizione dei due bit. Ogni bit potrebbe essere verificato individualmente attraverso le seguenti sei istruzioni:

```
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ANI     80H     ; Isola il bit 7
        JZ      PRD     ; Se il bit 7 è zero scavalca l'alimentazione del martelletto
LOP3    IN      2       ; Ingresso della Porta I/O 2 all'Accumulatore
        ANI     10H     ; Isola il bit 4
        JZ      LOP3    ; Attende per il valore non zero prima dell'Alimentaizone
```

Se HAMMER ENABLE è rivelato basso, l'esecuzione si ramifica all'istruzione con la label PRD. Si troverà che questa istruzione pone fine al programma all'inizio della sequenza d'istruzione che esegue **il ritardo di 2 millisecondi PRINTWHEEL READY.**

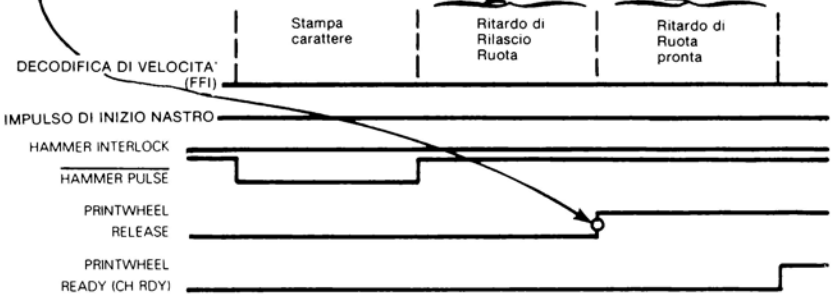
Si noti che la sequenza di cinque istruzioni illustrata in Figura 4–5 verifica $\overline{\text{HAMMER ENABLE}}$ basso all'interno dell'anello che verifica HAMMER INTERLOCK alto. Ora $\overline{\text{HAMMER ENABLE}}$ sarà alto o basso per la durata del ciclo di stampa; esso non cambierà livello durante il ciclo di stampa. Perciò il fatto che esso venga continuamente verificato è ridondante — questo non ha uno scopo ma non è dannoso.

Ora viene alimentato il martelletto. La sequenza d'istruzione che origina l'alimentazione del martelletto realizza le fasi da Ⓐ a Ⓘ che sono già state descritte. Allo scopo di rendere più facile la comprensione di questa sequenza d'istruzione, questa viene riportata di seguito aggiungendo le label da Ⓐ ad Ⓘ :

```
; Alimenta il martelletto di stampa
        IN      2       ; Pone HAMMER POULSE basso. Uscita 0
        ANI     FBH     ; Al bit 2 della Porta I/O 2
        OUT     2
   Ⓐ    IN      1       ; Ingresso all'Accumulatore del carattere ASCII
   Ⓑ    ANI     7FH     ; Maschera il bit di ordine elevato
   Ⓒ    SUI     20H     ; Sottrae 20H
        LXI     H,INDX  ; Carica l'indirizzo base della tabella con indice in HL
   Ⓓ    ADD     L       ; Somma i contenuti dell'Accumulatore ad HL
        MOV     L,A
   Ⓔ    MOV     A,M     ; Carica l'indice nell'Accumulatore
   Ⓕ    ADD     A       ; Moltiplica per 2
        LXI     H,DELY  ; Carica l'indirizzo base della tabella di ritardo in HL
   Ⓖ    ADD     L       ; Somma i contenuti dell'Accumulatore ad HL
        MOV     L,A
        MOV     E,M     ; Carica la costante di ritardo in D, E
   Ⓗ    INX     H
        MOV     D,M
LOP4    DCX     D       ; Esegue il ritardo lungo
        MOV     A,D
   Ⓘ    ORA     E
        JNZ     LOP4
        IN      2       ; Alla fine del ritardo fa uscire 1 al bit 2
        ORI     4       ; Della Porta I/O 2 (HAMMER PULSE ALTO)
        OUT     2
; Esegue il ritardo di tempo di 3ms per il rilascio della ruota
```

Viene ora seguito un ritardo di 3 millisecondi PRINTWHEEL RELEASE e la fine di questo ritardo di tempo è indicata dal segnale PRINTWHEEL RELEASE che esce alto. Successivamente viene eseguito il ritardo di 2 millisecondi PRINTWHEEL READY:

```
;ESEGUE UN RITARDO DI TEMPO DI RILASCIO RUOTA DI 3MS
        LXI     D,F7H   ;CARICA LA COSTANTE DI TEMPO IN D,E
LOP5    DCR     D       ;DECREMENTA D,E
        MOV     A,D     ;VERIFICA SE IN D,E C'È ZERO
        ORA     E
        JNZ     LOP5    ;RIDECREMENTA SE NON 0
;USCITA 1 AL BIT 0 DELLA PORTA I/O 2 QUESTO PONE
;ALTO PW REL
        IN      2       ;INGRESSO ALL'ACCUMULATORE DELLA PORTA I/O 2
        ORI     1       ;PONE AD 1 IL BIT 0
        OUT     2       ;USCITA RISULTATO
;ESEGUE UN RITARDO DI RUOTA PRONTA DI 2 MILLISECONDI
PRD     MVI     A,0     ;CARICA CON 0 L'ACCUMULATORE
LOP6    DCR     A       ;DECREMENTA A
        JNZ     LOP6    ;SE A NON È DECREMENTATO A 0, LO RIDECREMENTA
```



Prima del termine del ciclo di stampa indicato da PRINTWHEEL READY (CH READY) alto, il programma deve assicurarsi che il nastro non sia finito. Se $\overline{\text{EOR DET}}$ è rivelato basso il programma permane in un anello senza fine finchè il nastro non è stato cambiato; allora $\overline{\text{EOR DET}}$ entrerà alto dalla logica esterna.

Quando $\overline{\text{EOR DET}}$ e rivelato, le istruzioni finali del programma pongono alto PRINTWHEEL READY, così si ritorna all'inizio del programma e si attende il successivo ciclo di stampa.

ERRORI DELLA LOGICA DEL PROGRAMMA

Il programma sviluppato in questo capitolo contiene un errore logico che potrebbe non verificarsi in una realizzazione a logica digitale. L'errore è nel calcolo del ritardo di tempo dell'impulso del martelletto.

In una realizzazione a logica digitale, il codice ASCII per qualsiasi carattere sarebbe trattato come sette segnali distinti. Questi segnali potrebbero essere combinati in qualche modo per generare uno dei segnali di ritardo di tempo da H1 ad H6. **Non importa quale combinazione di codice ASCII entra, uno dei segnali del ritardo di tempo da H1 ad H6 uscirà alto;** se la logica di generazione del segnale non è errata, verrà originato il segnale di un ritardo di tempo, anche se può essere il segnale errato.

CONTROLLO DI LIMITE

Si osservi ora la realizzazione del programma in linguaggio assembly. E' abbastanza semplice consultare la tabella dell'Appendice A e vedere che i codici ASCII validi coprono il campo

da 20_{16} a $7A_{16}$. Questo non previene il progettista logico dall'impiego del sistema a microcalcolatore formante un sistema speciale che comprende caratteri non comuni, rappresentati da codici esterni al normale campo ASCII. Il programma potrebbe far uscire qualche risultato molto strano in queste circostanze. Si supponga che il codice ASCII 10_{16} sia stato adottato per rappresentare un carattere speciale. Allora il tentativo di consultare la tabella con indice caricherebbe nell'Accumulatore quello che si trova nel byte di memoria $n-10_{16}$.

Non si sa cosa potrebbe esserci in questo byte di memoria; con tutta probabilità questo byte sarà impiegato per immagazzinare un codice d'istruzione, forse un valore esadecimale a due digits. Si supponga che esso contenga $2A_{16}$: la fase successiva del programma duplicherà $2A_{16}$ aggiungendo all'indirizzo base della tabella di ritardo ed entrando al codice di ritardo iniziale dalla locazione di memoria $m+54_{16}$.

Data la configurazione del microcalcolatore illustrata in Figura 4-2, questa locazione di memoria potrebbe essere facilmente uno degli indirizzi duplicati che erroneamente accedono a qualche byte di memoria poichè è stata usata imprudentemente la logica di selezione del chip singolo. Impiegando la logica più complessa di selezione del chip, allora le variazioni non potrebbero tentare di accedere ad un byte di memoria che non esiste. Nel caso precedente, non si può dire quale lunghezza dell'impulso del martelletto venga generata; nel caso estremo potrebbe essere generato un impulso del martelletto estremamente lungo poichè si vorrebbe recuperare 0 da una locazione di memoria non esistente e questo valore potrebbe essere interpretato come la costante di ritardo iniziale per l'anello del programma di ritardo lungo. L'impulso del martelletto sarebbe lungo 720 millisecondi:

65,536 x 11 = 720,896 microseconds

Tempo in microsecondi per eseguire una volta l'anello di ritardo lungo

Risulta un anello di ritardo massimo poiché il valore iniziale 0000_{16} sarà decrementato e poi verificato

Ora allo scopo di evitare questo problema ci sono due possibilità:

1) **La logica del programma può semplicemente ignorare qualsiasi codice ASCII non valido.**
2) **La logica del programma può generare un impulso del martelletto di larghezza inadempiente per invalidare il codice ASCII.**

Se si ignorano i caratteri speciali la conclusione è ovvia: il sistema a microcalcolatore non può essere impiegato in qualsiasi applicazione che richieda caratteri speciali da stampare. Poichè il carattere speciale è ignorato non accadrà nulla quando tale codice di carattere è rivelato all'ingresso — non ci sarà impulso del martelletto, né movimento del carrello, né posizionamento.

Fornendo un impulso del martelletto inadempiente per i caratteri speciali, questi caratteri vanno stampati ma essi possono creare disuniformità di densità del testo stampato.

Caso per caso il progettista logico dovrà fare una scelta. Entrambe le sequenze d'istruzione possono essere inserite nel programma esistente come segue:

```
                -
                -
                OUT   2
                IN    1      ;INGRESSO ALL'ACCUMULATORE DEL CARATTERE ASCII
Controllo per   ANI   7FH    ;MASCHERA IL BIT DI ORDINE ELEVATO
codice ASCII    SUI   20H    ;SOTTRAE 20H
valido inserito qui
                LXI   H,INDX ;CARICA L'INDIRIZZO BASE DELLA TABELLA CON INDICE IN HL
                ADD   L      ;SOMMA AD HL I CONTENUTI DELL'ACCUMULATORE
                MOV   L,A
```

```
        MOV     A,M   ;CARICA L'INDICE DELL'ACCUMULATORE
        ADD     A     ;MOLTIPLICA PER 2
```

Ecco la sequenza d'istruzione che ignora i codici ASCII non standard:

```
        -
        -
        -
        IN      1       ; Ingresso del carattere ASCII all'Accumulatore
        ANI     7FH     ; Maschera il bit di ordine elevato
. Confronta il codice ASCII col valore più basso consentito
        CPI     20H
        JM      PRD     ; Se il codice è 1FH o minore, scavalca l'alimentazione del
                        ; martelletto
; Confronta il codice ASCII col valore più alto consentito
        CPI     7AH
        JP      PRD     ; Se il codice è 7BH o maggiore scavalca l'alimentazione del
                        ; martelletto
; Il codice ASCII è valido
        SUI     20H     ; Sottrae 20H
        LXI     H,INDEX ; Carica l'indirizzo base della tabella con indice ad HL
        -
        -
        -
```

La seconda scelta, illustrata di seguito, stampa un carattere non riconosciuto con una densità intermedia, impiegando il codice di densità 3:

```
        -
        -
        -
        IN      1       ; Ingresso del carattere ASCII all'Accumulatore
        ANI     7FH     ; Il bit di ordine elevato
; Confronta il codice col valore consentito più piccolo
        CPI     1FH
        JP      OK      ; Se il codice è 20H o maggiore verifica il limite superiore
; Il codice è illegale. Assume una densità pari a 3
NOK     MVI     A,6     ; Carica la densità doppia
        JMP     NEXT
; Confronta il codice ASCII col valore legale più grande
OK      CPI     7AH
        JP      NOK     ; Se il codice è 7BH o maggiore assume densità 3
. Il codice ASCII è valido
        SUI     20H     ; Sottrae 20H
        LXI     H,INDEX ; Carica l'indirizzo base della tabella con indice in HL
        ADD     L       ; Somma i contenuti dell'Accumulatore ad HL
        MOV     L,A
        MOV     A,M     ; Carica l'indice nell'Accumulatore
        ADD     A       ; Moltiplica per 2
NEXT    LXI     H,DELY  ; Carica l'indirizzo base della tabella di ritardo in HL
        -
        -
        -
```

Entrambe le sequenze d'istruzione di invalida del codice ASCII costituiscono una soluzione semplicistica del problema.

CONFRONTO IMMEDIATO

SALTO SU CONDIZIONE

L'unica nuova caratteristica introdotta è l'impiego dell'istruzione di Confronto Immediato (CPI). Questa istruzione sottrae i dati immediati nell'operando dai contenuti dell'Accumulatore. Il risultato dell'operazione è scartato, questo significa che i contenuti dell'Accumulatore non sono alterati; comunque i flags di stato riflettono il risultato della sottrazione. Si impiega l'istruzione JM (Salta Meno) per identificare un risultato negativo che significa che il dato

immediato nell'operando è maggiore del valore dell'Accumulatore. Analogamente una istruzione JP (Salta su Positivo) identifica un valore dell'operando immediato che è minore o uguale ai contenuti dell'Accumulatore.

TRAIETTORIE DI ESECUZIONE DI ISTRUZIONE CONDIZIONALE

Nella seconda sequenza di istruzione, se il valore dell'operando immediato è minore o uguale ai contenuti dell'Accumulatore, l'istruzione JP origina una diramazione ad una istruzione successiva con la label OK. Le traiettorie effettive di esecuzione del programma per la seconda sequenza di istruzione possono apparire confuse ad un non iniziato alla programmazine; perciò **si illustrano le traiettorie di esecuzione come segue:**

```
        IN      1       ; Ingresso all'Accumulatore del carattere ASCII
        ANI     7FH     ; Maschera il bit di ordine elevato
; Confronta il codice ASCII col valore legale più piccolo
        CPI     1FH
Ⓐ       JP      OK      ; Se il codice è 20H o magg. verifica il limite superiore
; Il codice è illegale. Assume una densità 3
NOK     MVI     A,6     ; Carica densità doppia
        JMP     NEXT
; Confronta il codice ASCII col valore legale più grande
OK      CPI     7AH
        JP      NOK     ; Se il codice è 7BH o maggiore assume densità 3
;Codice ASCII valido
        SUI     20H     ; Sottrae 20H
Ⓒ       LXI     H,INDEX ; Carica in HL l'indirizzo base della tabella con indice
        ADD     L       ; Somma ad HL i contenuti dell'Accumulatore
        MOV     L,A
        MOV     A,M     ; Carica l'indice nell'Accumulatore
        ADD     A       ; Moltiplica per 2
NEXT    LXI     H,DELY  ; Carica in HL l'indirizzo base della tabella di ritardo
```

Le traiettorie di esecuzione, sopra indicate mediante lettere cerchiate, possono essere interpretate come segue:

Ⓐ Un codice ASCII supera la prova "valore legale più basso" ma ora deve essere provato per "valore legale più alto".

Ⓑ Il codice ASCII è insufficiente nella prova "valore legale più basso". Il programma carica due volte la densità di inadempimento nell'Accumulatore e si dirama alla sequenza d'istruzione che accede la costante di ritardo opportuna per questa densità di inadempimento. Questo salto è illustrato da Ⓒ .

Ⓒ Un carattere che ha superato la prova "valore ASCII legale più basso" è successivamente controllato "per valore ASCII legale più alto"; se esso non supera questa prova allora l'esecuzione del programma si dirama, come mostrato da Ⓓ , alle istruzioni che assumono una densità di inadempienza di 3 . Ⓓ , infatti, consente Ⓑ .

Ⓓ Un carattere ASCII che supera entrambe le prove "valore legale più basso" e "valore legale più alto" è elaborato attraverso la traiettoria di istruzione Ⓔ . Le istruzioni di questa traiettoria caricano l'indice di densità appropriata nell'Accumulatore.

RESET ED INIZIALIZZAZIONE

Allo scopo di completare il programma si devono costruire le istruzioni necessarie per il reset e l'inizializzazione.

Le istruzioni di reset saranno eseguite ogni volta che è vero l'ingresso RESET al sistema a microcalcolatore.

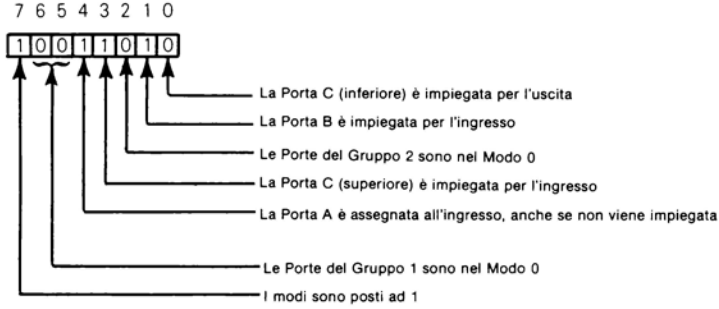
Le istruzioni di inizializzazione saranno eseguite ogni volta che il sistema è avviato.

Non c'è nessuna ragione perchè le sequenze d'istruzione di Reset ed Inizializzazione dovrebbero coincidere; in molte applicazioni possono essere richieste due sequenze di istruzione distinte e separate. **D'altra parte è abbastanza comune usare Reset al posto dell'inizializzazione del sistema.** Questo significa che quando si inizia ad alimentare il sistema, RESET è pulsato al livello vero; e questo fa partire l'intero sistema logico basato sul microcalcolatore.

Nel caso che si considera il programma di Reset è davvero semplice. Il tutto si riduce a far uscire un comando di controllo alla Porta I/O 3, la porta di controllo dell'8255 Interfaccia Periferiche Programmabile, quindi porre i segnali di uscita nella condizione "tra cicli di stampa". Il comando di controllo seleziona il Modo 0 con l'assegnazione della porta I/O appropriata. **Ecco la sequenza d'istruzione di inizializzazione necessaria:**

```
        ORG     0
; RESET ed inizializzazione del sistema
; Codice di controllo dell'uscita all'8255
; Interfaccia Periferica Programmabile
        MVI     A,9AH
        OUT     3
; Pone alto HAMMER PULSE, PW READY e
; PW REL, pone basso INIZIO MOVIMENTO DEL NASTRO
        MVI     A,7
        OUT     2
```

Ecco come è costruito il codice di controllo dell'8255 PPI:



SOMMARIO DEL PROGRAMMA

Prima di tutto, sarebbe una buona idea mettere assieme l'intero programma, come sviluppato in questo capitolo. Si comprenderanno le necessarie direttive Assembler. Questo programma finale è illustrato in Figura 4–6.

Ora che il programma è terminato, si noti che la memoria RAM non è stata impiegata. I registri della CPU hanno fornito una memoria di lettura/scrittura sufficiente a maneggiare tutti i dati variabili.

La memoria di programma ROM da 1 K byte è sufficiente a contenere l'intero programma più le due tabelle dei dati.

Eseguita la realizzazione su sistema a microcalcolatore all'interno dei confini della logica compresa in questo capitolo, si potrebbero ora eliminare i due chips di memoria RAM. Con tutta probabilità ci sarebbero numerose altre funzioni che potrebbero essere economicamente comprese nel sistema a microcalcolatore e queste richiederebbero quasi certamente la presenza di qualche memoria RAM. Ci sono nove byte di memoria di lettura/scrittura forniti dai 7 registri della CPU e dal puntatore dello Stack della CPU questi normalmente sono insufficienti per qualsiasi applicazione pratica.

```
INDEX EQU    0380H  ; Eguaglia l'indirizzo base della tabella con indice
DELY  EQU    03EEH  ; Eguaglia l'indirizzo base della tabella di ritardo - 2
      ORG    0
; Inizializzazione e reset del sistema
; Uscita del codice di controllo all'8255
; Interfaccia Periferica Programmabile
      MVI    9AH
      OUT    3
; Inizialmente pone alti l'impulso martelletto, PW READY
; E PW REL, pone basso INIZIO MOVIMENTO DEL NASTRO
      MVI    A,7
      OUT    2
; Programma del ciclo di stampa
; Tra cicli di stampa verifica FFI (bit 5 della Porta I/O 2 per un valore 0)
START IN     2      ; Ingresso all'Accumulatore della Porta I/O 2
      ANI    20H    ; Isola il bit 5
      JNZ    START  ; Se non 0, ritorno a START
; Inizializza il ciclo di stampa. Uscita 0 ai bit 0
; Ed 1 della Porta I/O 2. Uscita 1 al bit 3 della Porta I/O 2
      MVI    A,0CH  ; Carica la maschera nell'Accumulatore
      OUT    2      ; Uscita alla Porta I/O 2
; Uscita 0 al bit 3 della Porta I/O 2. Questo completa
; L'impulso d'INIZIO MOVIMENTO DEL NASTRO
      MVI    A,4    ; Carica la maschera nell'Accumulatore
      OUT    2      ; Uscita alla Porta I/O 2
; Verifica della fine del posizionamento della ruota
LOP1  IN     2      ; Ingresso all'Accumulatore della Porta I/O 2
      ANI    20H    ; Isola il bit 5
      JZ     LOP1   ; Se 0 ritorno a LOP1
; Esegue il ritardo di assestamento di 2ms della ruota
      MVI    A,0    ; Carica l'Accumulatore con 0
LOP2  DCR    A      ; Decrementa A
      JNZ    LOP2   ; Se A non è decrementato a 0, lo ri-decrementa
; Verifica le condizioni di alimentazione del martelletto
LOP3  IN     2      ; Ingresso all'Accumulatore della Porta I/O 2
      RLC           ; Muove il bit 7 nel Carry
      JNC    PRD    ; Se Carry è 0, scavalca l'alimentazione del martelletto
      ANI    20H    ; Isola il bit 4 che ora è il bit 5
      JZ     LOP3   ; Attende un valore non zero prima dell'alimentazione
; Alimenta il martelletto
      IN     2      ; Pone basso l'impulso del martelletto. Uscita 0 •
      ANI    FBH    ; Al bit 2 della Porta I/O 2
      OUT    2
      IN     1      ; Ingresso del carattere ASCII all'Accumulatore
      ANI    7FH    ; Maschera il bit di ordine elevato
```

Figura 4-6. Un semplice programma del ciclo di stampa

```
; Confronta il codice ASCII col valore legale più basso
        CPI     20H
        JM      PRD     ; Se il codice è 1FH o minore, scavalca l'alimentazione del
                        ; martelletto
; Confronta il codice ASCII con il valore legale più alto
        CPI     7AH
        JP      PRD     ; Se il codice è 7BH o più alto, scavalca l'alimentazione del
                        ; martelletto
; Il codice ASCII è valido
        SUI     20H     ; Sottrae 20H
        LXI     H,INDEX ; Carica in HL l'indirizzo base della tabella con indice
        ADD     L       ; Somma ad HL i contenuti dell'Accumulatore
        MOV     L,A
        MOV     A,M     ; Carica l'indice nell'Accumulatore
        ADD     A       ; Moltiplica per 2
        LXI     H,DELY  ; Carica in HL l'indirizzo base della tabella di ritardo
        ADD     L       ; Somma ad HL i contenuti dell'Accumulatore
        MOV     L,A
        MOV     E,M     ; Carica la costante di ritardo in D, E
        INX     H
        MOV     D,M
LOP4    DCX     D       ; Esegue un ritardo lungo
        MOV     A,D
        ORA     E
        JNZ     LOP4
        IN      2       ; Alla fine del ritardo uscita 1 al bit 2
        ORI     4       ; Della Porta I/O 2 (impulso del martelletto alto)
        OUT     2
; Esegue un ritardo di tempo di rilascio della ruota di 3ms
        LXI     D,F7H   ; Carica la costante di tempo in D, E
LOP5    DCR     D       ; Decrementa D, E
        MOV     A,D     ; Verifica per 0 in D, E
        ORA     E
        JNZ     LOP5    ; Ri-decrementa se non 0
                        ; Uscita 1 al bit 0 della Porta I/O 2. Questo pone alto PW REL
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     1       ; Pone ad 1 il bit 0
        OUT     2       ; Uscita risultato
; Esegue un ritardo di ruota pronta di 2ms
PRD     MVI     A,0     ; Carica con 0 l'Accumulatore
LOP6    DCR     A       ; Decrementa A
        JNZ     LOP6    ; Se non è decrementato a 0, lo ri-decrementa
                        ; Verifica per EOR DET (bit 6 della Porta I/O 2)
                        ; Uguale a 0 come pre-requisito per terminare il ciclo di stampa
LOP7    IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ANI     40H     ; Isola il bit 6
        JZ      LOP7    ; Ritorno e ri-verifica se non è 0
                        ; Alla fine del ciclo di stampa pone ad 1 il bit 1 della Porta
                        ; I/O 2. Questo pone alto CH RDY
        IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        ORI     2       ; Pone ad 1 il bit 1
        OUT     2       ; Pone ad 1 il bit 1
        JMP     START   ; Salta alla nuova verifica del ciclo di stampa
```

Figura 4-6. Un semplice programma del ciclo di stampa

; Esegue qui la tabella con indice

ORG 0380H

I dati rappresentati dell'indice 90 entrano qui di seguito. I dati compaiono nel campo dello mnemonico, un byte per riga.

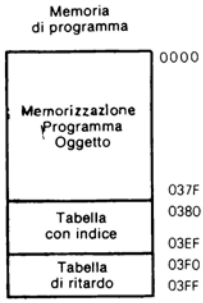
; Segue qui la tabella di ritardo

ORG 03F0H

Seguono qui i dati rappresentanti i 6 ritardi. I dati compaiono nel campo mnemonico, due byte per riga.

Figura 4–6. Un semplice programma del ciclo di stampa (segue).

Questa è la mappa finale della memoria di programma che identifica il modo in cui il programma illustrato in Figura 4–6 impiega la memoria ROM:



Capitolo 5
UNA PROSPETTIVA DEL PROGRAMMATORE

Il programma sviluppato al Capitolo 4 è considerevolmente più breve e più facile di quello della simulazione digitale del Capitolo 3. Mentre è stata percorsa una strada lunga al Capitolo 4 c'è ancora una strada da percorrere. Il programma della Figura 4-6 tratta la logica da realizzare come una singola funzione di trasferimento, ma esso non è un programma ben scritto.

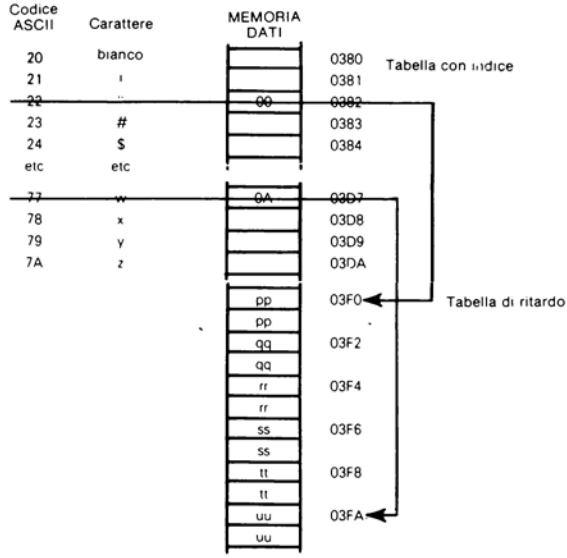
Al progettista di logica digitale una delle cose più confuse sulla programmazione è l'estrema facilità con cui si può fare la stessa cosa in dieci modi diversi. Questo implica che alcune realizzazioni sono più efficienti di altre? Infatti è così. Fino ad un limite elevato la scrittura di programmi efficienti è un talento, proprio come la creazione di logica digitale efficiente; ma ci sono certe regole che, se seguite, almeno aiuteranno ad evitare errori ovvi. In questo capitolo si considererà il programma originato al Capitolo 4 e si osserverà con un po' più di cura.

EFFICIENZA DELLA PROGRAMMAZIONE SEMPLICE

La prima cosa da fare, dopo la scrittura di un programma sorgente è di osservarlo nella ricerca di modi elementari per eliminare istruzioni.

CONSULTAZIONI DI TABELLA EFFICIENTE

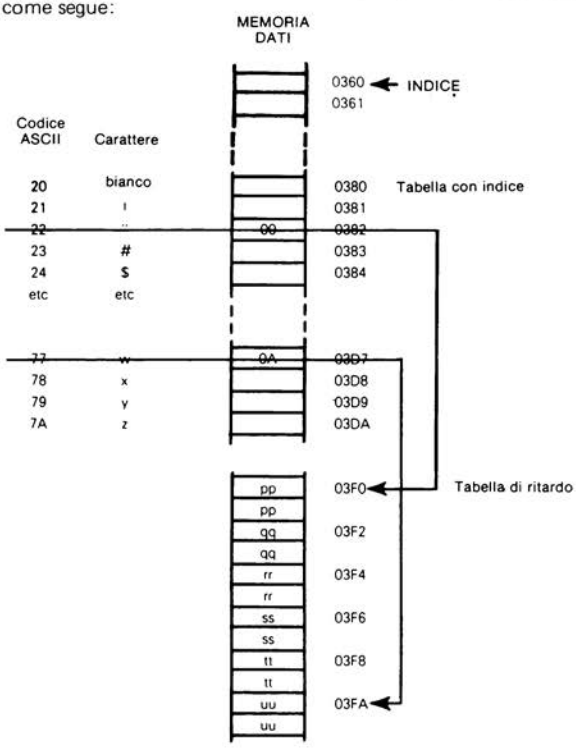
Mediamente si troverà che è possibile ridurre un programma a due terzi della sua lunghezza originale semplicemente riscrivendo le sequenze d'istruzione più efficienti. Nella Figura 4-6 l'esempio più ovvio di trascuratezza di programmazione coinvolge l'indice della Tabella. Il programma carica un valore tra 1 e 6 da un byte della Tabella con Indice quindi moltiplica questo valore per due prima di aggiungere l'indirizzo base della Tabella di Ritardo. **Perchè non si immagazzina direttamente il doppio dell'indice nella tabella con Indice?** Questo elimina un'istruzione come segue:



```
        —
        —
        —
; Il codice ASCII è valido
        SUI     20H     ; Sottrae 20H
        LXI     H,INDEX ; Carica l'indirizzo base della tabella con indice
        ADD     L       ; Somma i contenuti dell'Accumulatore ad HL
        MOV     L,A
        MOV     A,M     ; Carica l'indice X 2 nell'Accumulatore
        LXI     H,DELY  ; Carica l'indirizzo base della tabella di ritardo in HL
        ADD     L       ; Somma i contenuti dell'Accumulatore ad HL
        MOV     L,A
Istru-  —
zione   —
ADD     —
eliminata
```

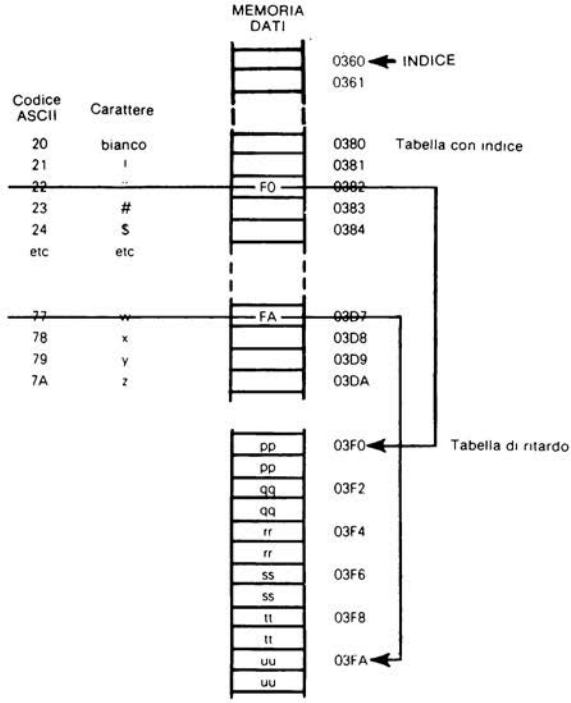
Nella precedente sequenza d'istruzione si noti che è stata rimossa l'istruzione seguente l'istruzione MOV su fondo scuro.

Esiste ancora un certo numero di modi per rendere la Tabella di Ritardo più efficiente da consultare. Per esempio **perchè sottrarre 20_{16} dal codice ASCII?** Se si somma il codice ASCII all'indirizzo base non c'è nulla da fermare come Uguaglianza di questo indirizzo base, rappresentato dal simbolo INDEX, ad un valore 20_{16} minore del primo byte effettivo della Tabella con Indice. La sequenza d'istruzione si riduce ulteriormente come segue:



```
INDEX EQU   0360H    ; Indice uguagliato all'indirizzo base della tabella 20H
      -
      -
      -
; Codice ASCII Valido
      LXI   H,INDEX  ; Carica l'indirizzo base della tabella con indice -20H
Istru-ADD   L        ; Somma i contenuti dell'Accumulatore ad HL
zione MOV   L,A
SUI   MOV   A,M      ; Carica l'indice X2 nell'Accumulatore
elimi-LXI   H,DELY   ; Carica l'indirizzo base della tabella di ritardo in HL
nata  ADD   L        ; Somma i contenuti dell'Accumulatore ad HL
      MOV   L,A
```

Così INDEX è ora eguagliata a 0360_{16} – che significa che non è più necessario sottrarre 20_{16} dal codice ASCII. E' stata eliminata l'istruzione SUI che precedeva l'istruzione LXI su fondo scuro. **Ora invece dell'immagazzinamento doppio dell'indice della densità di carattere nella Tabella con Indice perchè non immagazzinare la seconda metà dell'indirizzo della Tabella di Ritardo?** Il programma sarà contratto ulteriormente come segue:




```
INDEX EQU   0360H    ; Eguaglia l'indice all'indirizzo base della tabella -20H
      -
      -
      -
; Il codice ASCII è valido
      LXI   H,INDEX  ; Carica l'indirizzo base della tabella con indice -20H
      ADD   L        ; Somma i contenuti dell'Accumulatore ad HL
```

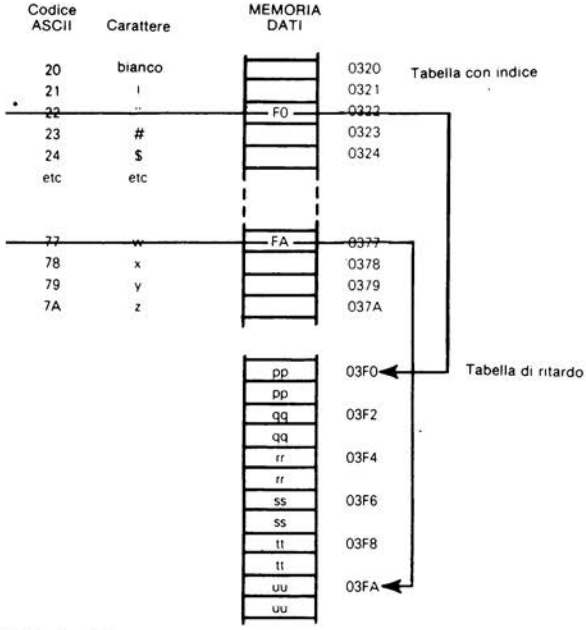
```
MOV  L,A
MOV  L,M      ; Carica il byte di basso ordine dell'indirizzo della tabella di
              ; ritardo
MVI  H,03H    ; Carica il byte di alto ordine dell'indirizzo della tabella di
              ; ritardo
```

Altre due istruzioni sono scomparse.

Sono state tolte quattro istruzioni dalla sequenza che carica la costante di ritardo iniziale dell'alimentazione del martelletto — e non è ancora tutto.

TABELLA POSIZIONATA PER SEMPLIFICARE LA SEQUENZA D'ISTRUZIONE DI ACCESSO

Perchè non muovere l'intera Tabella con Indice cosicchè invece di occupare le locazioni di memoria da 0380_{16} a $03DA_{16}$, essa occupi le locazioni di memoria da 0320_{16} a $037A_{16}$? Il codice ASCII, depurato del bit di parità, diviene ora il byte di basso ordine dell'indirizzo della Tabella con Indice e la sequenza d'istruzione si contrae ulteriormente come segue:



Il codice ASCII è valido

```
MVI   H,03H   ; Carica l'indirizzo della tabella con indice, byte di ordine alto
MOV   L,A     ; Muove il byte di basso ordine dell'indirizzo ad L
MOV   L.M     ; Carica il byte di basso ordine dell'indirizzo della tabella
              ; di ritardo
```

Si supponga di dover stampare un carattere "w". Prima dell'esecuzione della prima delle precedenti tre istruzioni l'Accumulatore contiene 77_{16} come risultato della precedente esecuzione delle istruzioni:

```
IN    1
ANI   7FH
```

In seguito all'esecuzione dell'istruzione:

```
MVI   H,03H
```

il registro H conterrà 03_{16}; questa è la metà superiore dell'indirizzo di memoria implicato. Successivamente l'istruzione:

```
MOV     L,A
```

muove 77_{16} dall'Accumulatore al registro L. H ed L ora contengono 0377_{16}; questo è l'effettivo indirizzo implicato. La successiva istruzione:

```
MOV     L,M
```

muove al registro L i contenuti del byte di memoria indirizzato da HL.

HL contiene 0377_{16}. Il byte di memoria 0377_{16} contiene FA_{16}, perciò FA_{16} è mosso al registro L. Il nuovo indirizzo implicato è $03FA_{16}$ e questo è l'indirizzo della Tabella di Ritardo richiesto.

Da nove istruzioni ci si è ricondotti a tre ed il solo prezzo pagato è di aver mosso la Tabella con Indice ad una nuova area della memoria dati.

Per assicurarsi della comprensione della modifica del programma si osservino le sequenze d'istruzione vecchia e nuova come mostrato di seguito fianco — a ← fianco, senza i campi di commento

```
Vecchio Programma                    Nuovo Programma

; Il codice ASCII è valido
        SUI     20H                  MVI     H,03H
        LXI     H,INDEX              MOV     L,A
        ADD     L                    MOV     L,M
        MOV     L,A
        MOV     A,M
        ADD     A
        LXI     H,DELY
        ADD     L
        MOV     L,A
```

Sfortunatamente non esistono regole d'oro che, se seguite, assicurano sempre la scrittura del programma più corto possibile. Una volta scritti alcuni programmi si comprenderà come lavorano le singole istruzioni e come a loro volta generano efficienza. Lo scopo delle pagine precedenti è stato quello di dimostrare l'enorme differenza tra un programma compatto ed uno diretto. Se si ha una produzione di volume elevato conviene ridurre le dimensioni del programma impiegando tempo e denaro — allora si può essere in grado di eliminare qualche chip ROM.

UTILIZZAZIONE HARDWARE

Tutti i programmatori di calcolatori cercano di scrivere programmi in linguaggio assembly efficienti. Comunque solo i programmatori di microcalcolatori devono considerare l'utilizzazione hardware come un contribuente integrale dell'efficienza della programmazione.

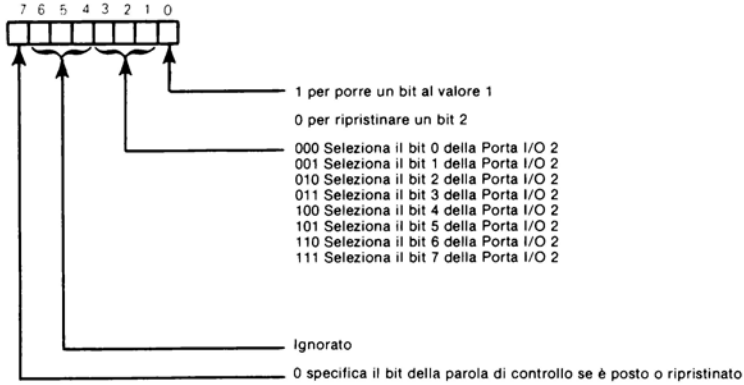
Ora è stato impiegato l'Interfaccia Periferica Programmabile 8255 nel Modo 0, accedendo ai bit della porta nel modo più ovvio. Si esplori qualche altra alternativa.

ISTRUZIONI SPECIFICHE-HARDWARE

ISTRUZIONI DI SET/RESET DI BIT

Si osservi che molte volte si posizionano e ripristinano singoli bit che divengono segnali d'ingresso e di uscita. L'8255 PPI ha una porta di Controllo I/O che nel caso che si considera è indirizzata come Porta I/O 3. I bit singoli della Porta I/O 2 possono essere posizionati o ripristinati facendo uscire un'appropriata parola di controllo alla Porta I/O 3. Que-

sto è il formato della parola di controllo:



ISTRUZIONI DIPENDENTI HARDWARE

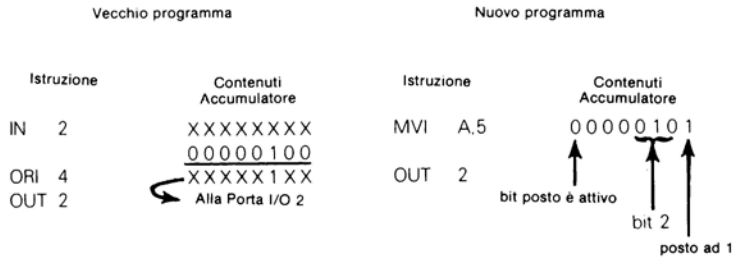
Si noti che questa istruzione di posizionamento/ripristino di bit è dipendente dall'hardware; essa fa assegnamento sulla logica interna all'8255 PPI per le caratteristiche dell'istruzione da realizzare.

Le istruzioni di controllo di set/reset di bit non risparmiano fasi di lettura dello stato del segnale d'ingresso; ma esse convertono le tre istruzioni in due istruzioni ogni volta che si vuole cambiare lo stato di un segnale d'uscita. Scorrendo il programma riassunto in Figura 4–6 ecco i cambiamenti effettivi:

```
        Vecchio Programma          Nuovo Programma
; Alimenta il martelletto. Pone basso l'impulso del martelletto esce 0 al bit 2 della
; Porta I/O 2.
        IN      2                  MVI     A,4
        ANI     FBH                OUT     3
        OUT     2                  –
        –                          –
        –                          –
; Alla fine del ritardo esce 1 al bit 2 della Porta I/O 2
; (Impulso del martelletto alto)
        IN      2                  MVI     A,5
        ORI     4                  OUT     3
        OUT     2                  –
        –                          –
        –                          –
; Uscita 1 al bit 0 della Porta I/O 2
; Questo pone alto PW REL
        IN      2                  MVI     A,1
        ORI     1                  OUT     3
        OUT     2                  –
        –                          –
        –                          –
; Alla fine del ciclo di stampa pone ad 1 il bit 1 della Porta I/O 2
; Questo pone alto CH RDY
        IN      2                  MVI     A,3
        ORI     2                  OUT     3
        OUT     2                  JMP     START
        JMP     START
```

SET/RESET DI BIT ILLUSTRATO

Nel caso si abbiano ancora dubbi su come opera il set/reset di bit, si illustrerà graficamente il divenire alto di HAMMER PULSE. Questo richiede l'uscita di un 1 al bit 2 della Porta I/O 2:

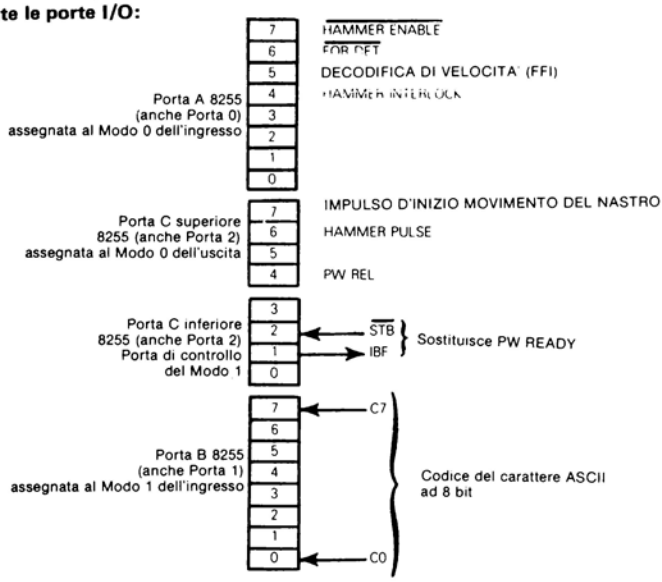


IMPIEGO DIRETTO DELLE CARATTERISTICHE HARDWARE

Si assumerà che la logica esterna impieghi il segnale PRINTWHEEL READY per assicurare che non si tenta di inviare un nuovo carattere finché il codice del carattere precedente non è stato elaborato. Si espanderanno alcune istruzioni posizionando basso PRINTWHEEL READY all'inizio del ciclo di stampa quindi ripristinandolo alto alla fine del ciclo di stampa.

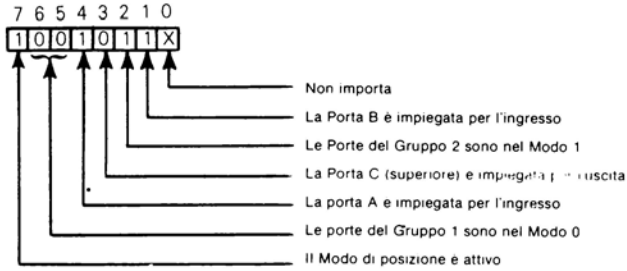
Senza una chiara definizione della logica esterna al sistema a microcalcolatore non c'è modo di conoscere se i segnali d'uscita PRINTWHEEL RELEASE e PRINTWHEEL READY sono richiesti esternamente per definire ritardi di tempo specifici o se sono semplicemente impiegati per assicurare che si consente ad un carattere di essere stampato completamente prima di tentare l'inizio della stampa del carattere successivo. Se la sola funzione del segnale PRINTWHEEL READY è di assicurare che non entri un nuovo carattere nel sistema a microcalcolatore prima che sia stato stampato il vecchio carattere, allora si può fare a meno del segnale PRINTWHEEL READY e sostituirlo con

l'ingresso con strobe, impiegando il Modo 1 per la Porta I/O 1. Ecco come saranno ora assegnate le porte I/O:

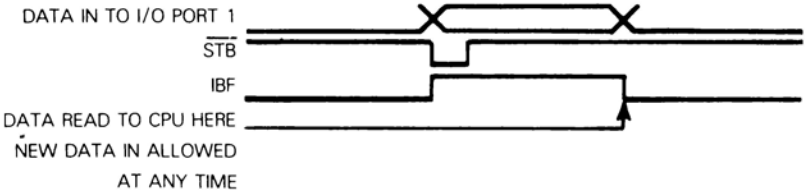


CODICE DI CONTROLLO PPI

Ecco come è costruito il codice di controllo dell'8255 PPI:



Quando la logica esterna fa entrare un nuovo carattere alla Porta I/O 1, essa simultaneamente fa entrare un segnale di strobe basso al pin 2 della Porta I/O 2 ($\overline{STB}$). In questo istante l'8255 PPI farà uscire IBF alto al pin 1 della Porta I/O 2. IBF rimarrà alto finchè non sono state eseguite le istruzioni per la lettura dei contenuti della Porta I/O 1 nella CPU. In questo istante IBF sarà ripristinato al livello basso. Questo può essere illustrato come segue:



Ora la logica del ciclo di stampa legge solo i contenuti della Porta I/O 1 una volta durante ogni ciclo di stampa. Nel Modo 1 la Porta I/O 1 è dotata di buffer; perciò tutta la logica esterna deve usare IBF come un segnale con strobe. Mentre IBF è basso la logica esterna è libera di fare entrare il carattere ASCII successivo. Se IBF è alto allora la logica esterna deve attendere.

Questo schema elimina il segnale PRINTWHEEL READY e qualsiasi istruzione richiesta per manipolare questo segnale.

Si noti che i segnali dei dati che sono usciti attraverso i bit da 0 a 3 della Porta I/O 2 sono stati mossi ai bit da 7 a 4 della Porta I/O 2. I segnali che erano allocati nei bit da 7 a 4 della Porta I/O 2 sono stati mossi alla Porta I/O 0. C'è una buona ragione di questo movimento. Si ricordi che i controlli di set/reset di bit lavorano solo per la Porta I/O 2. Se si vuole ridurre da 3 a 2 il numero di istruzioni richieste per cambiare lo stato di un segnale d'uscita, allora i segnali d'uscita devono essere assegnati ai pin della Porta I/O 2. Poichè non si può ridurre il numero delle istruzioni richieste per campionare i segnali d ingresso si deve appunto muovere questi segnali alla Porta I/O 0.

SUBROUTINES

Se si osserva ancora il programma della Figura 4–6 si noterà che in due punti all'interno di questo programma si eseguono le sequenze d'istruzione identiche a quelle per creare un ritardo di 2 millisecondi, così il fatto che queste tre istruzioni siano state ripetute non è molto grave. Se ci si pensa comunque esiste il problema potenzialmente per alcuni programmi più lunghi, di qualche utilizzazione di memoria molto antieconomica.

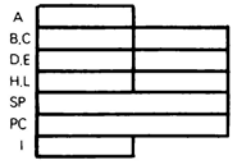
Nel Capitolo 4 è stato mantenuto semplice il programma perchè deve rimanere abbastanza piccolo da essere trattato su un libro; ma in sede di progetto si potrebbe avere una routine più complessa dove deve essere ripetuta una sequenza di 30 istruzioni piuttosto che una sequenza di 3 istruzioni. Si deve ora trovare qualche modo di includere la sequenza di istruzione una sola volta e poi di pervenire a questa sequenza singola da un certo numero di locazioni diverse all'interno del programma, come richiesto. Questo è quanto farà una subroutine.

Si considerino le 3 istruzioni che eseguono un ritardo di 2 millisecondi e si convertano in una subroutine. Questo è quanto accade a parti rilevanti del programma:

```
        ORG   0
        LXI   H,08FFH  ; Inizializza il puntatore dello stack alla fine
        SPHL           ; Dell'area dati
        –
        –
        –
; Esegue il ritardo di 2ms di sistemazione della ruota di stampa
        CALL  D2MS
        –
        –
        –
; Esegue il ritardo di 2 millisecondi di PRINTWHEEL READY
PRD     CALL  D2MS
        –
        –
        –
; Alla fine del ciclo di stampa pone ad 1 il bit 1 della Porta I/O 2
; Questo pone alto CH RDY
        MVI   A,3      ; Queste sono le nuove istruzioni
        OUT   3        ; Per porre un bit della Porta I/O 2
        JMP   START
```

```
; Subroutine per eseguire un ritardo di 2 millisecondi
D2MS    MVI    A,0     ; Carica con 0 l'Accumulatore
LOPD    DCR    A       ; Decrementa A
        JNZ    LOPD    ; Se A non è decrementato a 0, ri-decrementa
        RET            ; Ritorno dalla subroutine
```

Per comprendere come lavora la subroutine si assegnerà qualche indirizzo arbitrario di memoria al codice oggetto del programma sorgente; si mostrerà, fase per fase, cosa succede quando viene chiamata una subroutine ed al ritorno dalla subroutine. Prima di tutto **ecco la mappa di memoria che si considera:**



MEMORIA PROGRAMMA

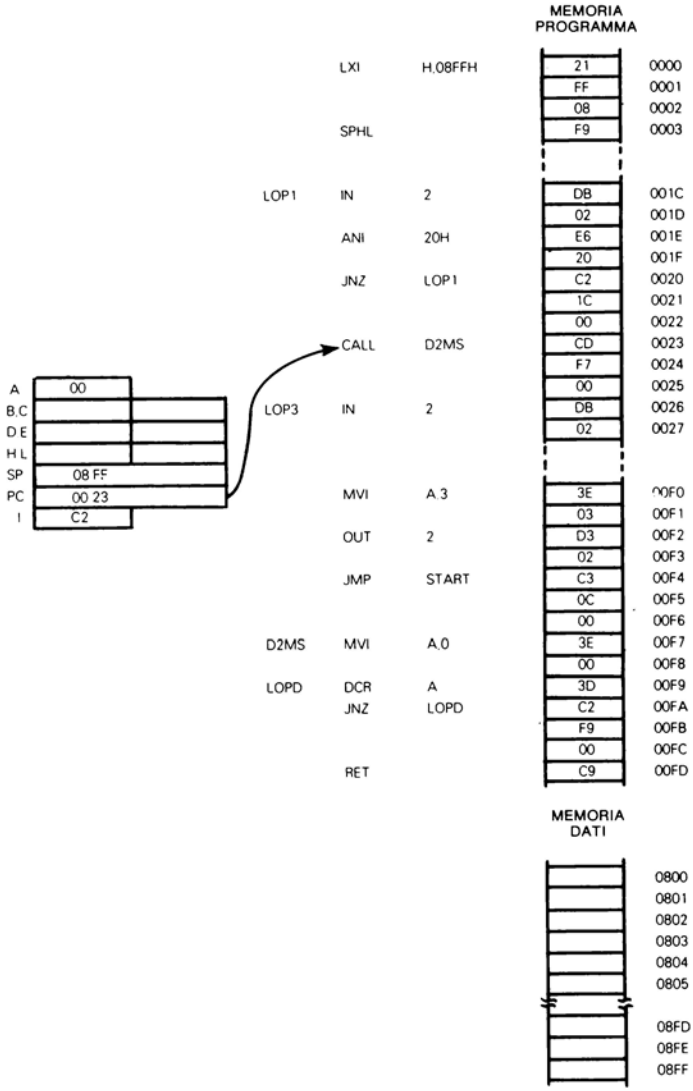
	LXI	H,08FFH	21	0000
			FF	0001
			08	0002
	SPHL		F9	0003
LOP1	IN	2	DB	001C
			02	001D
	ANI	20H	E6	001E
			20	001F
	JNZ	LOP1	C2	0020
			1C	0021
			00	0022
	CALL	D2MS	CD	0023
			F7	0024
			00	0025
LOP3	IN	2	DB	0026
			02	0027
	MVI	A,3	3E	00F0
			03	00F1
	OUT	2	D3	00F2
			02	00F3
	JMP	START	C3	00F4
			0C	00F5
			00	00F6
D2MS	MVI	A,0	3E	00F7
			00	00F8
LOPD	DCR	A	3D	00F9
	JNZ	LOPD	C2	00FA
			F9	00FB
			00	00FC
	RET		C9	00FD

MEMORIA DATI

	0800
	0801
	0802
	0803
	0804
	0805
	08FD
	08FE
	08FF

CHIAMATA DI SUBROUTINE

Si supponga di eseguire la prima istruzione CALL D2MS. A questo punti i registri conterranno i dati seguenti:



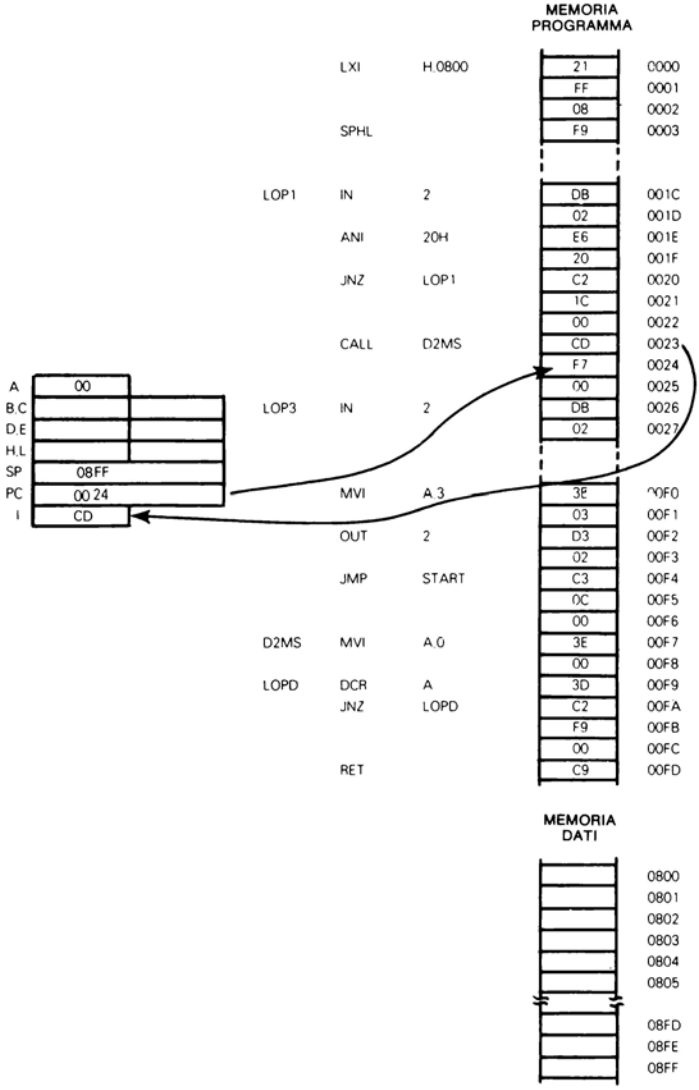
Il Contatore di Programma indirizza il primo byte del codice oggetto dell'istruzione Call, questo indirizzo è 0023_{16}. Il registro d'istruzione mantiene il codice oggetto per la istruzione eseguita più recentemente, questa è l'istruzione caricata al byte 0020_{16}. Come si noterà il Puntatore dello Stack è stato inizializzato all'inizio del programma; esso contiene $08FF_{16}$. In accordo con la Figura 4-2, questo è

l'indirizzo del primo byte della memoria di lettura/scrittura. Poichè lo stack non è stato impiegato, il puntatore dello stack conterrà $08FF_{16}$.

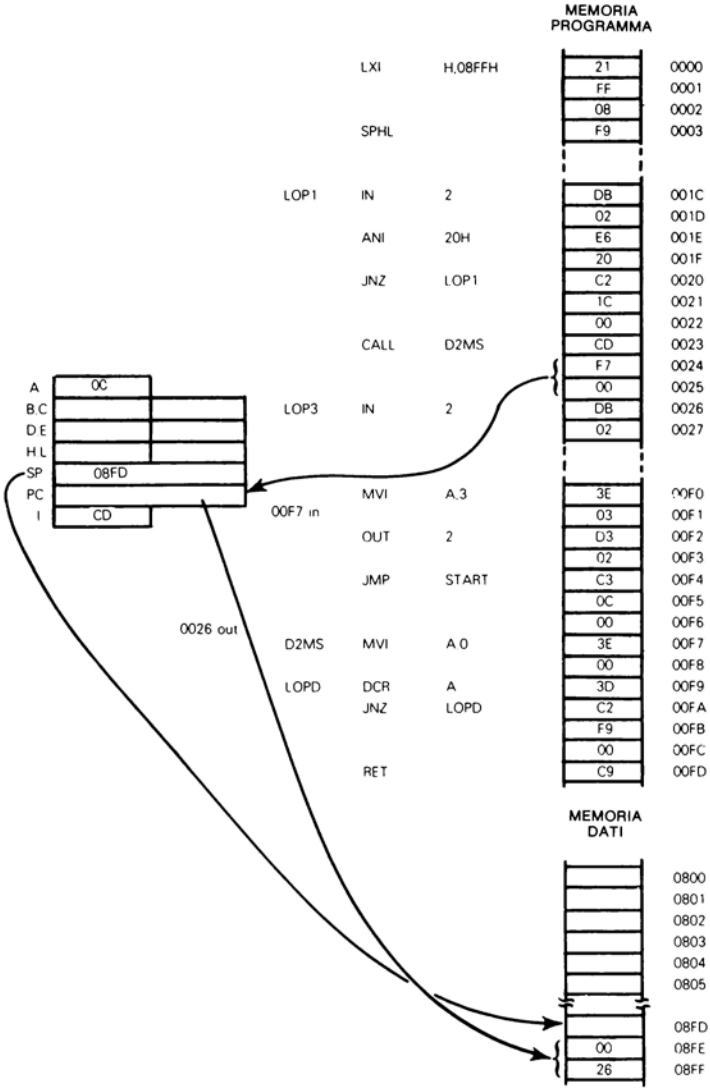
L'Accumulatore contiene 00 perchè questa era la condizione che ha originato l'esecuzione della rottura dell'anello di mantenimento iniziato a LOP1.

Ora quando è eseguita l'istruzione Call, si verificano le fasi seguenti:

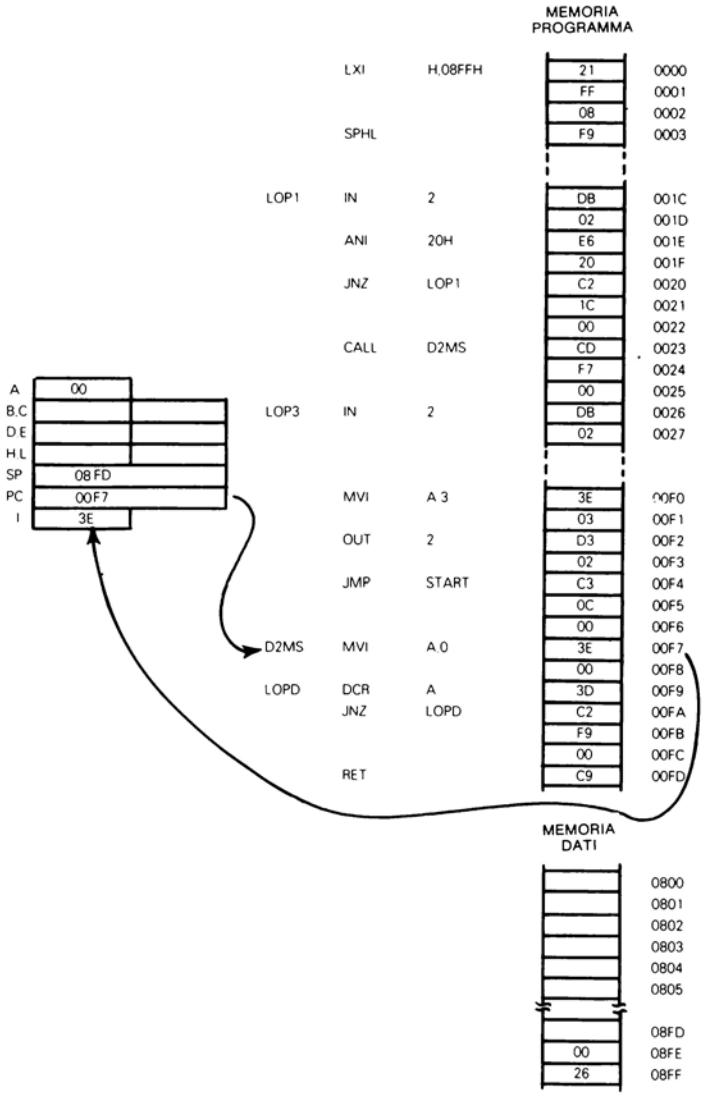
L'istruzione Call in codice oggetto è caricata nel registro d'Istruzione ed il Contatore di Programma è incrementato:



Il Contatore di Programma è incrementato di 2 scavalcando l'indirizzo di CALL. Questo valore incrementato è conservato nei primi due bytes dello stack. L'indirizzo di Call è poi caricato nel Contatore di Programma. Il Puntatore dello Stack è poi decrementato di 2 cosicchè esso è indirizzato al primo byte libero dello stack:



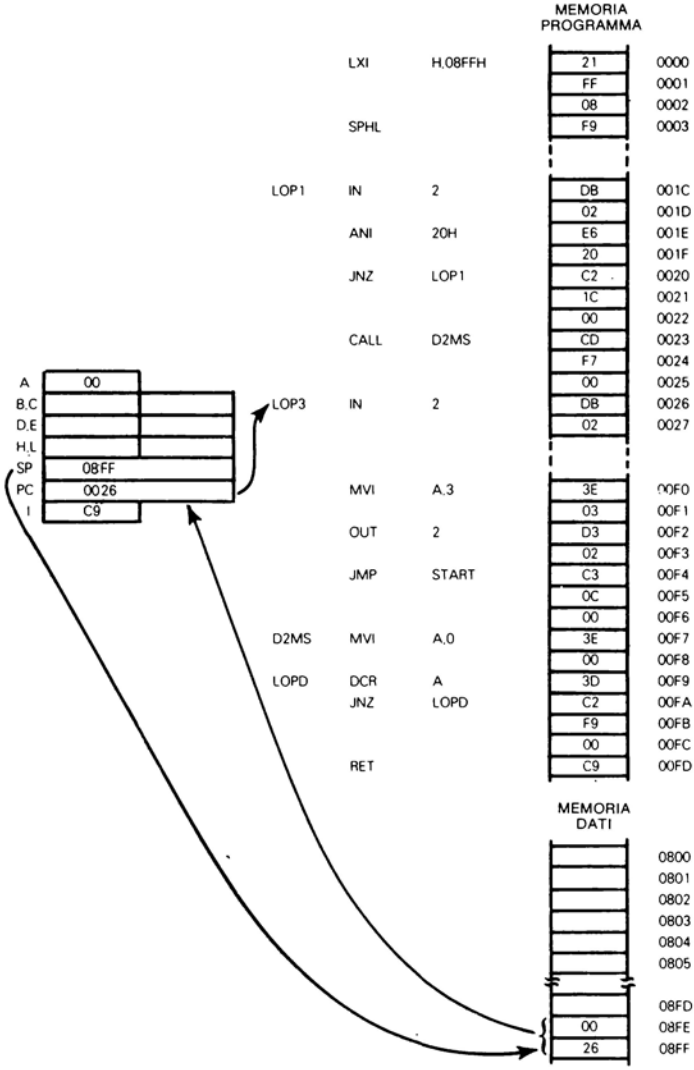
La successiva istruzione eseguita ha il suo codice oggetto immagazzinato nel byte di memoria $00F7_{16}$: questo è il byte di memoria ora indirizzato dal Contatore di Programma:



Le istruzioni all'interno dell'anello di ritardo di 2 millisecondi vengono ora eseguite ripetitivamente finchè i contenuti dell'Accumulatore decrementano da 01 a 00. Si ricordi, la prima volta l'Accumulatore è decrementato da 00_{16} ad FF_{16} e questo spiega perchè l'anello di 2 istruzioni cominciate a LOPD sarà eseguito 256 volte.

RITORNO DA SUBROUTINE

Quando infine l'Accumulatore decrementa da 01 a 00 l'esecuzione passa all'istruzione Return (RET). Questa istruzione incrementa di due i contenuti del Puntatore dello Stack, quindi muove i contenuti dei due bytes alla sommità dello Stack nel Contatore di Programma. Così l'esecuzione del Programma ritorna all'istruzione che segue la Call:



In conclusione accade questo:

Quando è stata eseguita l'istruzione Call, l'indirizzo dell'istruzione successiva è conservato nello Stack. L'istruzione Call fornisce l'indirizzo dell'istruzione successiva da eseguire.

L'istruzione successiva da eseguire è la prima della subroutine.

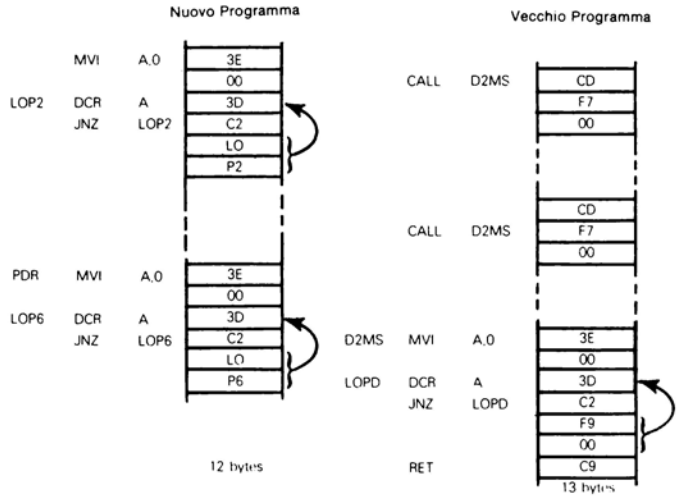
L'ultima istruzione della subroutine fa semplicemente in modo che l'indirizzo conservato alla sommità del puntatore dello stack ritorni al Contatore di Programma; questo, a sua volta, origina l'esecuzione della diramazione di ritorno all'istruzione seguente la Call.

QUANDO IMPIEGARE LE SUBROUTINES

C'è un prezzo associato con l'impiego delle subroutines:

1) **Ogni istruzione Call rappresenta tre bytes addizionali del codice oggetto.**
2) **La sequenza d'istruzione messa sotto forma di subroutine deve avere un'istruzione di Return che costa un byte del codice oggetto.**

Si osservi prima il caso specifico. Le tre istruzioni che costituiscono la subrotuine costituiscono un ritardo di 2 millisecondi ed occupano 6 byte del codice oggetto. Queste tre istruzioni si verificano tre volte; perciò combinate esse occupano 12 bytes del codice oggetto. Quando si passa ad una subroutine, l'aggiunta dell'istruzione Return aumenta da 6 a 7 i bytes del codice oggetto. Inoltre ci sono due istruzioni Call ed ognuna richiede tre bytes del codice oggetto — che significa che due istruzioni Call più la subroutine generano 13 bytes del codice oggetto. Questo può essere illustrato come segue:



Nel caso specifico, perciò, trasferendo la sequenza d'istruzione del ritardo di 2 millisecondi ad una subroutine costa un byte del codice oggetto. Questo ci costa quattro byte addizionali del codice oggetto — richiesti per inizializzare il Puntatore dello Stack; ed il sistema a microcalcolatore richiederà ora la memoria RAM.

Uno stack può esistere solo se è presente la memoria di lettura/scrittura.

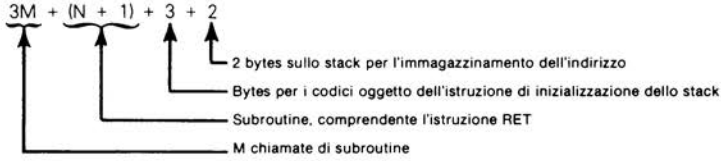
Ora questi commenti non implicano che la subroutine è una caratteristica della programmazione, da usare con moderazione; al contrario è difficile concepire qualsiasi programma che, quando ben scritto, non comprenda qualche subroutine. Ma si tenga ben presente **che c'è una dimensione minima di subroutine al di sotto della quale le subroutines in generale divengono antieconomiche.**

Si supponga che ci siano N byte del codice oggetto in una sequenza d'istruzione che si sta pianificando di convertire in una subroutine.

Si supponga che gli **N byte del codice oggetto si verifichino M volte;** questo significa che N byte del codice oggetto divengono una subroutine, essi saranno chiamati dall'istruzione CALL.

Senza subroutine saranno consumati MXN byte ripetendo N byte M volte.

Con le subroutines il numero di byte consumato è:



La subroutine è conveniente se 3M+N+6 è minore di MxN.

La Tabella 5-1 mostra la minima lunghezza conveniente della subroutine in funzione del numero di chiamate di subroutine.

Tabella 5-1. La lunghezza di subroutine conveniente più corta in funzione del numero di volte di chiamata della subroutine stessa

Numero di chiamate di Subroutine (M)	Lunghezza minima conveniente di Subroutine (N)
2	12 Bytes
3	8 Bytes
4	6 Bytes
5	6 Bytes
10	4 Bytes
20	4 Bytes

RITORNO CONDIZIONALE DA SUBROUTINE

Anche se nessuna delle sequenze d'istruzione ripetute all'interno del programma della Figura 4-6 sono abbastanza lunghe da giustificare la commutazione in subroutine si esplorerà ulteriormente l'impiego delle subroutine.

Proprio come esistono le istruzioni di salto condizionale, che si usano spesso all'interno di un anello di ritardo di tempo, così esistono istruzioni di chiamata di subroutine condizionali ed istruzioni di Ritorno da subroutine condizionali.

Le istruzioni Call e Return di subroutine condizionali sono particolarmente pratiche in subroutine lunghe all'interno delle quali ci sono percorsi di esecuzione variabili.

Si consideri la sequenza d'istruzione di alimentazione del martelletto della Figura 4-6. Dato il programma illustrato questa sequenza d'istruzione si verifica una sola volta, che significa che la conversione in una subroutine non avrebbe senso. **E' possibile immaginare un programma più estensivo che esegue una grande quantità di operazioni d'interfaccia, come la logica di alimentazione del martelletto che devé essere fatta scattare da un certo numero di ragioni diverse. Poichè la logica di alimentazione del martelletto consiste di un set abbastanza lungo, potrebbe essere assolutamente ingiuntivo il mettere queste istruzioni in una subroutine. Si consideri la seguente realizzazione di subroutine:**

```
; Subroutine di alimentazione del martelletto
PFIR  IN    2     ; Ingresso della Porta I/O 2 all'Accumulatore
      RLC         ; Muove il bit 7 in Carry
      RNC         ; Se Carry non è 1 ritorno
```

```
        ANI     20H     ; Isola il bit 4 che ora è il bit 5
        RZ              ; Se zero ritorno
Alimenta il martelletto
        MVI     A4      ; Pone basso HAMMER PULSE. Uscita 0
        OUT     3       ; Al bit 2 della Porta I/O 2
        IN      1       ; Ingresso del codice del carattere ASCII all'Accumulatore
        ANI     7FH     ; Maschera il bit di ordine elevato
; Confronta il codice ASCII con il valore legale più basso
        CPI     20H
        RM              ; Se il codice è 1FH o minore, scavalca l'alimentazione del
                        ; martelletto
; Confronta il codice ASCII con il valore legale più alto
        CPI     7AH
        RP              ; Se il codice è 7BH o maggiore scavalca l'alimentazione
; Codice ASCII valido
        MVI     H,03H   ; Indirizzo della tabella con indice, il byte di ordine elevato
        MOV     L,A     ; Muove il byte di basso ordine di indirizzo ad L
        MOV     L.M     ; Carica il byte di basso ordine dell'indirizzo della tabella
                        ; di ritardo
        CALL    LDLY    ; Chiama la subroutine di ritardo lungo
        MVI     A,5     ; Alla fine del ritardo esce 1 al bit 2 della
        OUT     3       ; Porta I/O 2 (HAMMER PULSE alto)
; Esegue un ritardo di tempo di 2ms per PRINTWHEEL RELEASE
        LXI     H,MS3
        CALL    LDLY
; Uscita 1 al bit 0 della Porta I/O 2.
; Questo pone PW REL alto
        MVI     A,1
        OUT     3
        RET             ; Ritorno da subroutine
; Subroutine di ritardo lungo. Assume H ed L
; Indirizzati al primo dei due byte dati che contengono
; La costante di ritardo iniziale
LDLY    MOV     E,M     ; Carica la costante di ritardo iniziale
        iNX     H
        MOV     D,M
LDLP    DCX     D       ; Esegue il ritardo lungo
        MOV     A,D
        ORA     E
        JNZ     LDLP
        RET             ; Ritorno alla fine del ritardo lungo
MS3     00F7H           ; Costante del ritardo di tempo di PRINTWHEEL RELEASE
```

RITORNO CONDIZIONALE

La subroutine precedentemente illustrata alimenta il martelletto solo se tutte le condizioni necessarie sono soddisfatte; un'uscita veloce è eseguita se nessuna condizione di alimentazione è soddisfatta. Le istruzioni di Ritorno condizionale sono su fondo scuro.

Si noti che sono state impiegate le sequenze d'istruzione più compatte per far uscire singoli bit alla Porta I/O 2 e per identificare il ritardo corretto di alimentazione del martelletto.

SUBROUTINE IDENTIFICATE

Si noti che **è stata inserita una subroutine all'interno di una altra subroutine. La sequenza d'istruzione del ritardo lungo è stata mossa ad una subroutine la cui prima istruzione ha la label LDLY.**

PARAMETRO DELLA SUBROUTINE

Questo porta a dire che le "subroutine sono nidificate". Una nuova caratteristica della subroutine LDLY è che essa richiede che la costante di ritardi iniziale sia immagazzinata in due bytes della memoria, il primo dei quali è indirizzato nei registri H ed L quando viene chiamata LDLY. **Le istruzioni all'interno della**

subroutine LDLY caricheranno effettivamente la costante di ritardo iniziale nei registri D ed E. La costante di ritardo iniziale diviene un parametro, che consente ad una subroutine di realizzare uno spettro completo di ritardi di tempo. I parametri della subroutine sono una caratteristica molto importante dell'impiego della subroutine.

La subroutine LDLY è chiamata una seconda volta, invece del caricamento della costante di tempo iniziale ($F7_{16}$) nei registri D ed E, si carica un indirizzo rappresentato dal simbolo MS3 nei registri H ed L. Il simbolo MS3 diverrà l'indirizzo dei due byte dati, dovunque in memoria; all'interno di questi due byte dati deve essere immagazzinato il valore $00F7_{16}$.

RITORNI MULTIPLI DA SUBROUTINE

La subroutine PFIR non è così pratica come potrebbe essere. Ci sono quattro ritorni condizionali da questa subroutine, ognuno dei quali è fatto scattare da una condizione di zero diversa. C'è anche un ritorno di subroutine seguente un'alimentazione del martelletto.

Come fa il programma chiamante a conoscere se il martelletto era o non era alimentato dopo che è stata chiamata PFIR? La convalida degli stati non è molto sicura perchè non può essere certo cosa accade alle condizioni dello stato durante l'esecuzione delle istruzioni di alimentazione del martelletto stesso. Non si può impiegare lo stato Carry per determinare se l'istruzione RNC è la condizione di ritorno che causa un'uscita dalla subroutine PFIR; questo perchè non si può dire cosa succede allo stato Carry mentre viene eseguito il resto della subroutine. Per esempio lo stato Carry sarà 0 se l'istruzione di ritorno condizionale origina un'uscita dalla subroutine PFIR.

Le subroutine che contengono un grande numero di uscite di errore condizionale, oltre al ritorno standard, spesso conterranno la logica che fa ritornare ad un numero di istruzioni diverse del programma chiamante. Si consideri il caso della subroutine PFIR. La sequenza d'istruzione che chiama questa subroutine può apparire come segue:

```
        –
        –
        –
RT0     CALL    PFIR    ; Chiama la subroutine di alimentazione del martelletto
        JMP     RT1     ; Ritorna qui per il riposizionamento della ruota
        JMP     RT0     ; Ritorna qui per HAMMER INTERLOCK basso
        JMP     RT2     ; Ritorna qui per codice ASCII minore di 20H
        JMP     RT3     ; Ritorna qui per codice ASCII maggiore di 7AH
; Le istruzioni che seguono sono eseguite dopo l'esecuzione
. Dell'alimentazione valida del martelletto
        –
        –
        –
; Le istruzioni che seguono sono eseguite per il riposizionamento
; Della ruota di stampa
RT1
        –
        –
        –
; Le istruzioni che seguono sono eseguite per codice ASCII minore di 20H
RT2
        –
        –
        –
  Le istruzioni che seguono sono eseguite per codice ASCII maggiore di 7AH
RT3
```

Ora per questo schema di lavoro la subroutine PFIR deve incrementare l'indirizzo di ritorno, che è immagazzinato alla sommità dello stack, in due byte, ogni volta è eseguito un ritorno condizionale. La subroutine PFIR è perciò modificata come segue:

```
; Subroutine di alimentazione del martelletto
PRIF IN      2      ; Ingresso della Porta I/O 2 all'Accumulatore
     RLC             ; Muove il bit 7 in Carry
     RNC             ; Ritorno se Carry non è 1
     CALL    INCR    ; Incrementa l'indirizzo di ritorno
     ANI     20H     ; Isola il bit 4 che è ora il bit 5
     RZ              ; Se zero ritorno
     CALL    INCR    ; Incrementa l'indirizzo di ritorno
; Alimenta martelletto
     MVI     A,4     ; Pone basso l'impulso del martelletto. Uscita 0
     OUT     3       ; Al bit 2 della Porta I/O 2
     IN      1       ; Ingresso all'Accumulatore del codice del carattere ASCII
     ANI     7FH     ; Maschera il bit di ordine elevato
; Confronta il codice ASCII col valore legale più basso
     CPI     20H
     RM              ; Se il codice è 1FH o minore scalvalca l'alimentazione del
                     ; martelletto
     CALL    INCR    ; Incrementa l'indirizzo di ritorno
; Confronta il codice ASCII con il valore legale più alto
     CPI     7AH
     RP              ; Se il codice è 7BH o maggiore scavalca l'alimentazione del
                     ; martelletto
     CALL    INCR    ; Incrementa l'indirizzo di ritorno
; Il codice ASCII è valido
     MVI     H,03H   ; Carica l'indirizzo della tabella con indice.
                     ; Il byte di ordine elevato
     MOV     L,A     ; Muove il byte di indirizzo di basso ordine ad L
     MOV     L,M     ; Carica il byte di basso ordine dell'indirizzo della tabella
                     ; di ritardo
     CALL    LDLY    ; Chiama la subroutine di ritardo lungo
     MVI     A,5     ; Alla fine del ritardo esce 1 al bit 2 della
     OUT     3       ; Porta I/O 2 (HAMMER PULSE alto)
; Esegue un ritardo di tempo di 3ms di rilascio della ruota
     LXI     H,MS3
     CALL    LDLY
; Uscita 1 al bit 0 della Porta I/O 2. Questo pone alto PW REL
     MVI     A,1
     OUT     3
     RET             ; Ritorno da subroutine
; Subroutine di ritardo lungo. Assume H ed L come indirizzo del primo dei due byte
; dati che mantengono costante il ritardo iniziale
LDLY MOV     E,M     ; Carica la costante di ritardo iniziale
     INX     H
     MOV     D,M
LDLP DCX     D       ; Esegue il ritardo lungo
     MOV     A,D
     ORA     E
     JNZ     LDLP
     RET             ; Ritorno alla fine del ritardo lungo
; Subroutine per incrementare l'indirizzo di ritorno della subroutine chiamante
INCR INX     SP      ; Incrementa due volte il puntatore dello stack
     INX     SP      ; Per accedere a PFIR indirizzo di ritorno
```

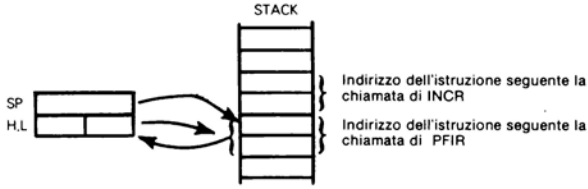
```
XTHL          ; Scambia HL con l'indirizzo di ritorno PFIR
INX     H     ; Somma 3 all'indirizzo di ritorno
INX     H
INX     H
XTHL          ; Ri-immagazzina l'indirizzo di ritorno
DCX     SP    ; Decrementa due volte il puntatore dello stack
DCX     SP
RET           ; Ritorno
```

MANIPOLAZIONE DELLO STACK

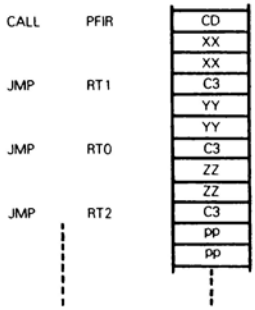
La subroutine INCR è interessante; essa mostra come può essere manipolato lo stack. Si dia uno sguardo a cosa succede.

Non appena entra la subroutine INCR i contenuti del Puntatore dello Stack sono aumentati di due. Questo ha l'effetto dell'indirizzo di ritorno PFIR piuttosto che l'indirizzo di ritorno INCR:

L'istruzione XTHL conserva semplicemente i contenuti dei registri H ed L e questi sono ora alla sommità dello Stack, mentre il movimento che era alla sommità dello stack va ai registri H ed L:



Le successive tre istruzioni aggiungono 3 ai contenuti dei registri H ed L, che ora conservano l'indirizzo di ritorno PFIR. Si somma tre all'indirizzo di ritorno, perchè se si osserva la sequenza chiamante, segue una serie di istruzioni di salto. Ogni istruzione di salto occupa tre byte, che significa che ogni volta che si scavalca il Ritorno Condizionale si deve incrementa di 3 l'indirizzo di ritorno:



La successiva istruzione XTHL ri-immagazzina semplicemente il Puntatore dello Stack ai suoi contenuti originali, cosicchè l'istruzione di Ritorno INCR preleverà l'indirizzo di ritorno corretto.

CHIAMATE DI SUBROUTINE CONDIZIONALI

Si vuole ora costruire un'altra subroutine che alimenti il martelletto ma non faccia alcuna prova per assicurarsi se il martelletto è alimentato. Questa subroutine assume semplicemente che nell'Accumulatore ci sia un carattere valido e che il martelletto deve essere alimentato. Tutta la logica, per determinare se l'alimentazione del martelletto è valida, è esterna alla subroutine di alimentazione del martelletto; perciò questa subroutine è chiamata condizionatamente non appena tutte le condizioni di alimentazione del martelletto sono soddisfatte. Ecco come appare il nuovo programma:

```
        –
        –
        –
; Verifica delle condizioni di alimentazione del martelletto
LOP3    IN      2       ; Ingresso all'Accumulatore della Porta I/O 2
        RLC             ; Muove il bit 7 nel Carry
        JNC     PRD     ; Se Carry è 0, scavalca l'alimentazione del martelletto
        ANI     20H     ; Isola il bit 4 che ora è il bit 5
        JZ      LOP3    ; Attende un valore non zero prima dell'alimentazione
; Ingresso del carattere da stampare
        IN      1       ; Ingresso all'Accumulatore del carattere ASCII
        ANI     7FH     ; Maschera il bit di ordine elevato
; Confronta il codice ASCII col valore legale più basso
        CPI     20H
        JM      PRD     ; Se il codice è 1F o minore scavalca l'alimentazione del
                        ; martelletto
; Confronta il codice ASCII col valore legale più alto
        CPI     7BH     ; Se il codice è minore di 7BH chiama
        CM      FIRE    ; La subroutine di alimentazione del martelletto
; Esegue il ritardo di 2 millisecondi di PRINTWHEEL READY
PRD     MVI     A,0     ; Carica con 0 l'Accumulatore
        –
        –
        –
```

Si noti che l'istruzione di Ritorno Condizionale riflette la logica di programmazione dell'OR, mentre l'istruzione di Chiamata Condizionale riflette la logica dell'AND. Così la subroutine PFIR comprende un certo numero di istruzioni di Ritorno Condizionale, ognuno dei quali sarà eseguito fornendo una qualsiasi delle condizioni di zero incontrate. La subroutine FIRE, d'altra parte, è chiamata condizionatamente solo quando è stata verificata l'ultima delle condizioni necessarie.

La subroutine FIRE non è mostrata in dettaglio poichè la sua scrittura aggiungerebbe poco alla comprensione dell'istruzione di Chiamata Condizionale. Con riferimento alla Figura 4-6, la subroutine FIRE consisterà di istruzioni per:

Porre basso il segnale dell'impulso del martelletto
Eseguire il ritardo dell'impulso di alimentazione del martelletto
Porre alto l'impulso del martelletto
Eseguire il ritardo di tempo di 3 millisecondi di sistemazione della ruota
Uscita alta di PW REL

MACROS

Trattando le subroutine si è omessa una considerazione — il programmatore. Le subroutine hanno un valore addizionale in questo se si può ridurre il numero di istruzioni del programma sorgente allora si ridurrà anche il tempo impiegato per la scrittura del programma sorgente essendo questo tempo proporzionale alla lunghezza del programma.

Si osservi ancora la subroutine del ritardo di tempo di 2 millisecondi. Sebbene il programma, in forma di subroutine, richieda più byte di codice oggetto, esso non richiede più istruzioni:

Vecchio Programma

```
        MVI     A,0
LOP2    DCR     A
        JNZ     LOP2
        -

        -
PDR     MVI     A,0
LOP6    DCR     A
        JNZ     LOP6
```

6 istruzioni
(12 byte)

Nuovo Programma

```
        CALL    D2MS
        -
        CALL    D2MS

        -
D2MS    MVI     A,0
LOPD    DCR     A
        JNZ     LOPD
        RET
```

6 istruzioni
(13 byte escludendo le istruzioni di stack ed inizializzazione)

Le subroutine possono diminuire la lunghezza del programma sorgente, aumentare la lunghezza del programma oggetto ed il tempo di esecuzione del programma.

Le Macros diminuiscono la lunghezza del programma sorgente ma non hanno assolutamente alcun effetto sul programma oggetto.

COS'E' UNA MACRO?

Una macro è una forma di programmazione "stenografica"; essa consente di definire una sequenza d'istruzione con un singolo mnemonico.

DEFINIZIONE DI MACRO

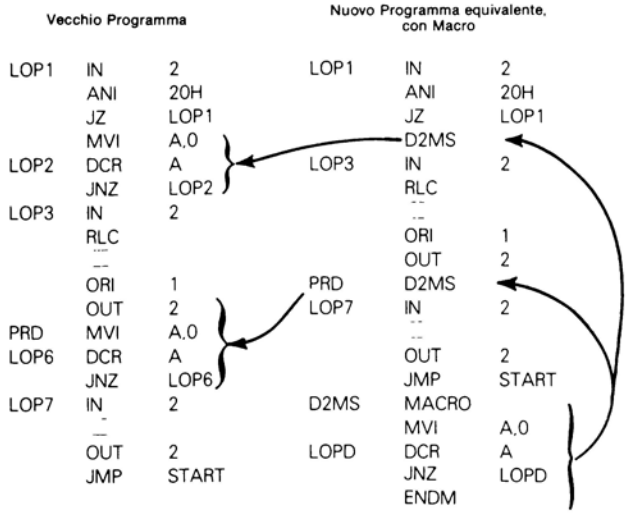
Si consideri la sequenza d'istruzione del ritardo di tempo di 2 millisecondi: questa si può definire come una macro, con la label D2MS, come segue:

```
D2MS    MACRO
        MVI     A,0
LOPD    DCR     A
        JNZ     LOPD
        ENDM
```

DIRETTIVE DELL'ASSEMBLATORE MACRO

Le due precedenti istruzioni su fondo scuro sono in realtà, direttive dell'assemblatore; esse sostengono una sequenza di istruzioni che in seguito possono essere identificate come un gruppo, impiegando la label della direttiva dell'assemblatore MACRO.

Ecco come si potrebbe impiegare il ritardo di tempo di 2 millisecondi del programma del ciclo di stampa:



Quando l'Assemblatore incontra il simbolo D2MS nel campo mnemonico esso sostituisce questo simbolo con le istruzioni sostenute delle direttive MACRO ed ENDM. L'Assemblatore conosce cosa impiega la macro, nell'eventualità che il programma contenga più di una macro, poichè il simbolo nel campo mnemonico deve essere identico alla label della direttiva MACRO.

Si noti che l'Accumulatore può eseguire anche un certo governo associato con l'impiego delle macros. Il "Vecchio Programma" sopra illustrato ha le label LOP2 e LOP6 per le due istruzioni DCR. Il "Nuovo Programma" ha una singola label, LOPD, dentro la macro. L'Assemblatore sa che una label che compare all'interno della definizione di una macro deve diventare una serie di label separate quando successivamente la macro è inserito un certo numero di volte nel programma sorgente.

LOCAZIONE DI DEFINIZIONE MACRO IN UN PROGRAMMA SORGENTE

Per riassumere si consideri semplicemente una sequenza di istruzioni ripetute, chiuse all'interno delle direttive MACRO ed ENDM, allora si danno le direttive delle macro con un'unica label. Ora si impiega la label della MACRO come se essa fosse un mnemonico di istruzione. La definizione della macro deve apparire una ed una sola volta, dovunque nel programma sorgente. E' un buon metodo raccogliere tutte le macro ed inserirle all'inizio, o alla fine, dell'intero programma sorgente.

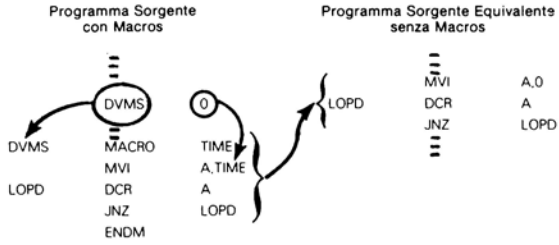
MACRO CON PARAMETRI

Le istruzioni all'interno di una macro possono avere operandi variabili; per esempio si può originare una macro di ritardo di tempo come segue:

```
DVMS    MACRO   TIME
        MVI     A,TIME
LOPD    DCR     A
        JNZ     LOPD
        ENDM
```

I simboli che compaiono nel campo operando della direttiva MACRO sono assunti dall'Assemblatore essere dei simboli di schermo; il riferimento della macro nel corpo del programma sorgente deve coinvolgere un campo dell'operando equivalente. Lo Assemblatore uguaglierà il campo dell'operando del riferimento al campo dell'operando della direttiva MACRO e farà le sostituzioni in accordo.

Ecco come viene operata la sostituzione:



Questo è un altro esempio; il riferimento macro:

```
      DVMS    80H
```

e equivalente a:

```
      MVI     A,80H
LOPD  DCR     A
      JNZ     LOPD
```

In funzione di quali Assemblatori si sta impiegando si può far giocare un ruolo interessante alla lista dei parametri macro; in teoria (ma non sempre, in pratica) non ci sono restrizioni alla lunghezza od alla natura della lista dei parametri della macro. Si supponga di voler variare il registro impiegato in una sequenza d'istruzione di un ritardo di tempo; con alcuni assemblatori si opererà come segue:

```
      DVMS    C,3CH
DVMS  MACRO   X,TIME
      MVI     X,TIME
LOPD  DCR     X
      JNZ     LOPD
      ENDM
```

L'Assemblatore sostituirà

```
      DVMC    C,3CH
```

con

```
      MVI     C,3CH
LOPD  DCR     C
      JNZ     LOPD
```

Comunque si dovrà consultare il manuale dell'Assemblatore che accompagna il sistema di sviluppo per conoscere le esatte caratteristiche per la macro.

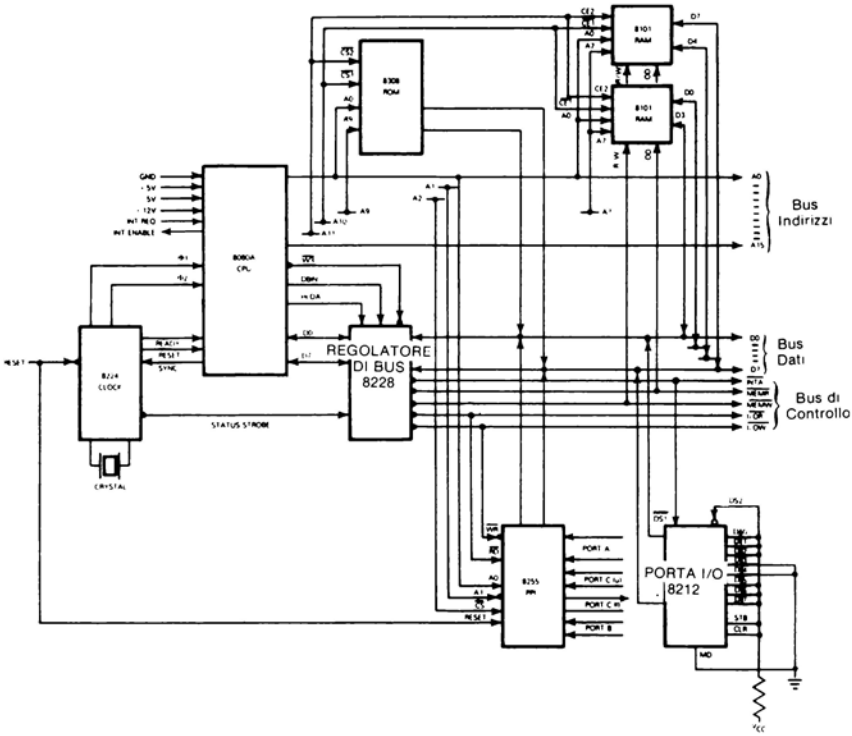


Figura 5-1. Configurazione del microcalcolatore con un interrupt singolo

INTERRUPTS

Potrebbe essere difficile giustificare l'inclusione di interrupts all'interno del sistema a microcalcolatore sviluppato al Capitolo 4. Infatti gli interrupts sarebbero impiegati abbastanza moderatamente nelle applicazioni di microcalcolatori.

QUANDO IMPIEGARE GLI INTERRUPTS

Non si entrerà in una lunga discussione sui vantaggi e svantaggi di interrupts all'interno del sistema a microcalcolatore, questo argomento è stato adeguatamente trattato in "An Introduction to Microcomputers", Volume I. Per riassumere, comunque, si ricorda che gli interrupts sono un valido strumento all'interno dei sistemi a microcalcolatore solo quando si stanno trattando eventi veloci ed asincroni.

Ora, avendo rilasciato un avvertimento contro l'impiego indiscriminato di interrupts, si procederà a considerare l'elaborazione dell'interrupt semplice nel programma del microcalcolatore nell'intento di dimostrare come esso è fatto.

CONSIDERAZIONI HARDWARE SULL'INTERRUPT

Per elaborare un interrupt all'interno di un sistema a microcalcolatore occorre fare entrare un segnale di richiesta di interrupt al livello alto alla CPU nell'istante in cui gli interrupts sono stati abilitati.

ABILITAZIONE DI INTERRUPT

Gli interrupts sono abilitati, o disabilitati dall'esecuzione delle istruzioni EI e DI, rispettivamente. La condizione di abilitato è identificata da un'uscita alta dal segnale INT ENABLE della CPU 8080 (INTE). La logica esterna non deve interrogare questo segnale prima della richiesta di un interrupt, qualsiasi richiesta di interrupt sarà semplicemente ignorata dalla CPU mentre gli interrupts sono stati disabilitati.

RICONOSCIMENTO DI INTERRUPT

ISTRUZIONE DI RESTART

Se una richiesta di interrupt è ricevuta mentre gli interrupts sono stati abilitati, allora dopo aver completato l'esecuzione dell'istruzione in corso, la CPU farà uscire un segnale di riconoscimento interrupt attraverso il Controllore di Sistema 8228. La logica esterna che sta richiedendo l'interrupt deve rispondere al segnale di riconoscimento facendo entrare un codice d'istruzione ad 8-bit che deve essere interpretato come un codice di istruzione da eseguire successivamente. Normalmente sarà considerata una delle otto possibili istruzioni di Restart. Queste istruzioni sono equivalenti alle chiamate di subroutine a byte singolo; esse fanno in modo che i contenuti del contatore di Programma vengano spinti nello Stack, dopodichè l'esecuzione del programma continua ad un indirizzo di memoria basso che può essere calcolato come segue:

RST N codice istruzione:	1 1 1 X X X 1 1	
	0 0 0	N = 0
	0 0 1	N = 1
	0 1 0	N = 2
	0 1 1	N = 3
	1 0 0	N = 4
	1 0 1	N = 5
	1 1 0	N = 6
	1 1 1	N = 7
Nuovo Programma Contenuti contatore:	0 0 0 0 0 0 0 0 0 0 X X X 0 0 0	

Perciò le istruzioni RST N sono equivalenti alle istruzioni CALL di subroutine, con le diramazioni all'esecuzione del programma come segue:

	Subroutine
RST 0	dirama a 0000_{16}
RST 1	dirama a 0008_{16}
RST 2	dirama a 0010_{16}
RST 3	dirama a 0018_{16}
RST 4	dirama a 0020_{16}
RST 5	dirama a 0028_{16}
RST 6	dirama a 0030_{16}
RST 7	dirama a 0038_{16}

COSTRUZIONE DEL CODICE D'ISTRUZIONE RST

Il modo più semplice per la costruzione di un'istruzione RST opportuna esternamente è attraverso una Porta I/O 8212. La logica necessaria è illustrata in Figura 5-1.

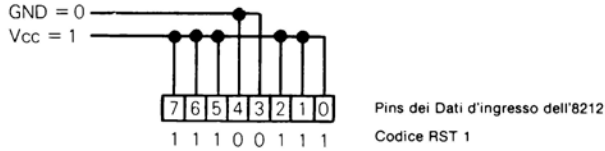
Ora nella Figura 5-1, la Porta I/O 8212 è stata impiegata nel modo più elementare possibile.

CONFIGURAZIONE DI INTERRUPT SINGOLO

Si esamini l'impiego di questa semplice porta I/O. Poichè c'è un interrupt, esso è collegato direttamente all'ingresso INT REQ della CPU 8080. L'uscita IN ENABLE CPU è semplicemente ignorata. Quando la sorgente logica esterna che può richiedere un interrupt fa entrare INT REQ al livello alto, allo istante successivo la CPU riconoscerà l'interrupt attraverso il Controllore di Bus 8228, facendo uscire basso $\overline{\text{INTA}}$ sul Bus di Controllo. Ora nella Porta I/O 8212 il segnale INTA è impiegato come uno dei segnali di selezione del dispositivo. L'altro segnale di selezione del dispositivo, DS2, deve essere alto; esso è perciò collegato a V_{CC}.

LA PORTA I/O 8212 IMPIEGATA IN UN SISTEMA D'INTERRUPT

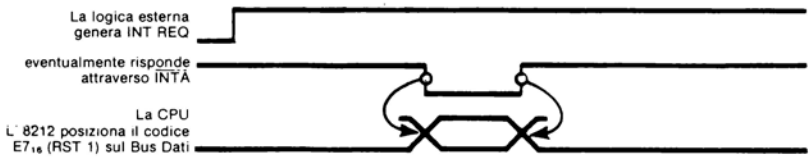
Il solo scopo della Porta I/O 8212 nella Figura 5-1 è di fare entrare l'istruzione RST 1 una volta che viene riconosciuto un interrupt mediante l'uscita di INTA al livello alto. Si seleziona arbitrariamente l'istruzione RST 1. **La sola istruzione RST che non può essere selezionata è RST 0, poichè la locazione di memoria 0 è impiegata eseguendo un reset.** Il codice dell'istruzione RST 1 è generato collegando i pins appropriati dei dati d'ingresso a massa ed al livello 1 effettivo:



Il pin modo (MD) è collegato a massa poichè la Porta I/O 8212 deve essere impiegata nel modo d'ingresso.

Poichè il solo criterio per fare uscire dati dalla Porta I/O 8212 è che $\overline{\text{INTA}}$ sia basso, gli ingressi STROBE e CLEAR sono disabilitati collegandoli a V_{CC}.

Riassumendo ecco cosa succede quando la logica esterna richiede un'interrupt:



Un progettista logico od un programmatore non ha bisogno di occuparsi del timing del Bus Dati. Il segnale $\overline{\text{INTA}}$ fa da strobe corretto al codice di istruzione RST 1 nella

CPU, facendo in modo che l'uscita della Porta I/O 8212 sia interpretata come un codice di istruzione, piuttosto che come dati.

Anche un breve esame del modo in cui la Porta I/O 8212 è stata incorporata nel sistema a microcalcolatore mostra che c'è abbondanza di modi più elaborati in cui la Porta I/O 8212 potrebbe essere impiegata; per esempio essa può manipolare interrupts multipli.

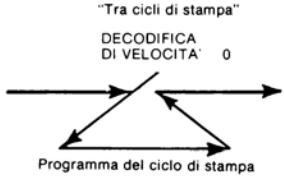
Il collegamento della richiesta di interrupt esterna direttamente alla CPU è un metodo primitivo di manipolazione di interrupt, ma abbastanza adeguato in un sistema a microcalcolatore dove solo un dispositivo esterno è probabilmente il richiedente di interrupt.

Prima di osservare alcuni degli schemi più elaborati di richiesta di interrupt si faranno considerazioni di programmazione associate con il semplice schema di Figura 5-1.

PROGRAMMA DI SERVIZIO INTERRUPT

Prima di tutto, **perchè si impiega un interrupt?**

Si potrebbe assumere che il sistema a microcalcolatore sia impiegato per fare più di una realizzazione logica di un ciclo di stampa. **Si supponga che ci sia una grande quantità di routine di governo logico richieste dall'interfaccia della stampante, con il risultato che l'intero ciclo di stampa può essere osservato su un evento intermittente asincrono. Ora, invece di avere il programma eseguito nel ciclo di istruzione "tra cicli di stampa", si assumerà che qualche altro programma sia continuamente eseguito tra cicli di stampa. L'esecuzione del programma del ciclo di stampa è fatta scattare dal segnale DECODIFICA DI VELOCITA' il cui complemento è collegato ad INT REQ nella Figura 5-1. Questo è lo schema di esecuzione di istruzione che ne risulta:**



Ecco la sequenza d'istruzione richiesta per eseguire il programma del ciclo di stampa che segue un interrupt derivato dall'inverso di decodifica di velocità:

```
        ORG    8
; Origine della routine di servizio interrupt del programma del ciclo di stampa
; A 0008H, poichè questo è l'indirizzo di esecuzione
; Che segue l'esecuzione di un'istruzione RST 1
        CALL   START ; Chiama come una subroutine il programma del ciclo di
                     ; spampa
        RET          ; Ritorno da interrupt
        ORG    NNNN
; Seleziona qualsiasi origine percorribile per il programma del ciclo di stampa
; Inizializza il ciclo di stampa. Uscita 0 ai bit 0 ed 1 della Porta I/O 2
; Uscita 1 al bit 3 della Porta I/O 2
START   MVI    A,0CH ; Carica la maschera nell'Accumulatore
        OUT    2     ; Uscita alla Porta I/O 2
        —
        —
        —
; Alla fine del ciclo di stampa, pone ad 1 il bit 1 della Porta I/O 2
; Questo pone alto CH RDY
        MVI    A,3
        OUT    3
        RET
```

Si noti che sono state rimosse le istruzioni "tra cicli di stampa"; START ora identifica la prima istruzione del ciclo di stampa stesso.

ORIGINE DEL PROGRAMMA DI INTERRUPT

L'origine specificata per il programma del ciclo di stampa è importante. Non si conosce quali altri programmi siano eseguiti all'interno del sistema a microcalcolatore o dove questi altri programmi possono risiedere nella memoria di programma: perciò non si può assegnare spazio di memoria al programma del ciclo di stampa in questo momento. Quando effettivamente si realizza l'intero sistema microcalcolatore si deve attentamente disegnare in quali esatte posizioni di memoria risiede ogni programma, ma per gli scopi dell'attuale illustrazione, questa è una considerazione completamente non importante. Si noti che le istruzioni finali del ciclo di stampa impiegano le istruzioni di controllo per porre un bit per modificare CH RDY; ma impiegando la Porta I/O 8255, la Porta 1 nel Modo 1 è stata assunta senza strobe.

L'istruzione finale JMP START è sostituita dalla semplice istruzione RETURN poichè l'intero programma del ciclo di stampa era in effetti richiamato come una subroutine.

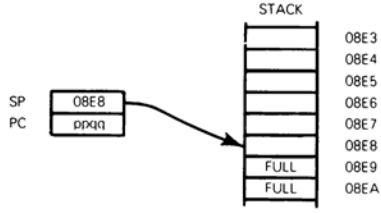
Il sistema a microcalcolatore insegue sé stesso all'interno della memoria eseguendo un interrupt che impiega lo stack. Viene di seguito spiegato come ciò è realizzato.

SOMMITA' DELLO STACK

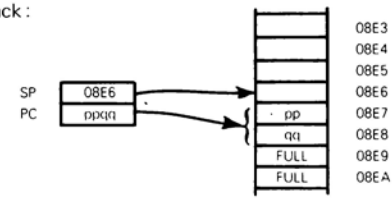
Quando precedentemente è stato osservato lo stack, esso è stato originato a $08FF_{16}$; questa era la sommità della memoria di lettura/scrittura e non si è entrati nello stack. Si assumerà ora che qualsiasi programma eseguito tra cicli di stampa, acceda allo stack, cosicchè quando è stato eseguito il ciclo di stampa il Puntatore dello Stack contiene l'indirizzo $08E8_{16}$. Si assumerà semplicemente che ci sia stato qualche livello della attività dello stack, ma la cosa non si conosce nè importa.

Seguendo il riconoscimento di interrupt, ecco come è impiegato lo stack:

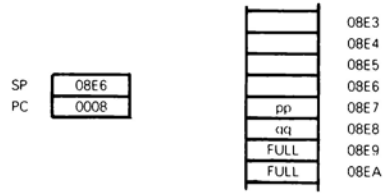
1) Quando il segnale di DECODIFICA DI VELOCITA' richiede un interrupt la situazione è questa:



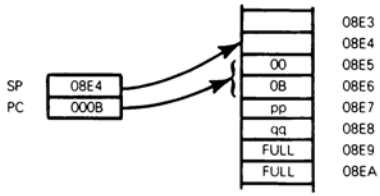
2) L'interrupt è riconosciuto. Prima i contenuti del Contatore di Programma sono conservati nello stack:



3) Poi l'istruzione RST 1 fa in modo che 0008 sia caricato nel contatore di Programma.

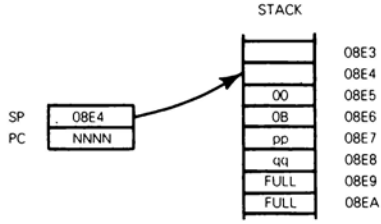


4) Alla locazione di memoria 0008 c'è l'istruzione CALL START. Questa fa in modo che l'istruzione successiva sia conservata nello stack:

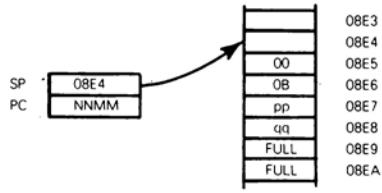


Si ricordi che l'istruzione successiva (RET) è caricata in $000B_{16}$ perchè l'istruzione CALL START occupa tre bytes 0008_{16}, 0009_{16} e $000A_{16}$.

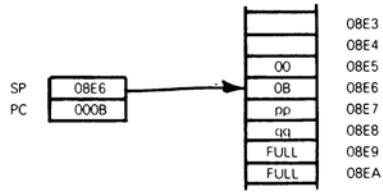
5) Ora l'indirizzo dell'istruzione con la label START è caricato nel Contatore di Programma. Questa è la prima istruzione della routine del ciclo di stampa:



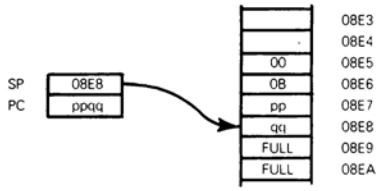
6) Si assuma che l'istruzione RET finale nella routine del ciclo di stampa sia immagazzinata nella locazione di memoria NNMM. Quando questa istruzione RET sta per essere eseguita, la situazione è questa:



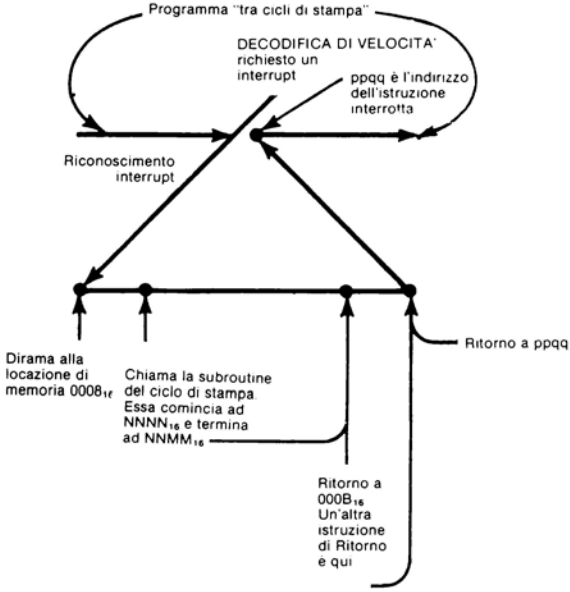
7) Quando viene eseguita l'istruzione RET, il Puntatore dello Stack è incrementato di 2, ed i due byte di sommità dello Stack sono mossi nel Contatore di Programma:



8) Un'altra istruzione RET è immagazzinata nella locazione di memoria 000B, cosicchè succede ancora la stessa cosa:



L'indirizzo dell'istruzione interrotta è stato ri-immagazzinato. Ecco un'illustrazione completa:



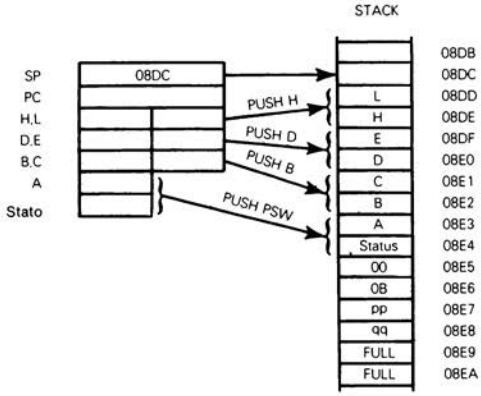
CONSERVAZIONE DEI REGISTRI E DELLO STATO

Il metodo appena descritto per l'elaborazione di un interrupt è stato mantenuto semplice per non generare fastidi in seguito al corso dell'esecuzione del programma; ma il programma non lavorerà. E' stato mostrato un programma di base interrotto allo scopo di eseguire la routine del ciclo di stampa; ma quando sarà interrotto il programma di base? Si ricordi che il programma interrotto condivide la stessa CPU e gli stessi registri del programma del ciclo di stampa. Occorre assumere che il programma interrotto abbia l'informazione utile memorizzata nei registri e forse i flag di stato hanno un significato che deve essere preservato. Dato il programma di servizio interrupt illustrato in generale, quando si ritorna dal programma del ciclo di stampa al programma interrotto, si stanno dando i contenuti di qualsiasi registro arbitrario del programma interrotto al programma del ciclo di stampa già terminato. Questo non si farà mai. **Si deve perciò sostenere il programma di esecuzione del ciclo di stampa con istruzioni che conservano i contenuti dei registri e dello stato nello stack — prima di modificare un singolo**

registro o stato; alla fine del programma i contenuti dei registri originale e di stato devono essere ri-immagazzinati. Ecco come appare a questo punto il programma del ciclo di stampa:

```
        ORG     8
; La routine di servizio interrupt del programma del ciclo di
; Stampa inizia a 008H, poichè questo è l'indirizzo di esecuzione
; Che segue l'esecuzione di un'istruzione RST 1
        CALL    START ; Chiama il programma del ciclo di stampa
                      ; Come una subroutine
        RET           ; Ritorno da interrupt
        ORG     NNNN
; Spinge i contenuti di tutti i registri e lo stato nello Stack
START   PUSH    PSW   ; Conserva l'Accumulatore e lo stato
        PUSH    B     ; Conserva B e C
        PUSH    D     ; Conserva D ed E
        PUSH    H     ; Conserva H ed L
; Seleziona qualsiasi origine percorribile per il programma del ciclo di stampa
; Inizializza il ciclo di stampa. Uscita 0 ai bit 0 ed 1 della
; Porta I/O 2. Uscita al bit 3 della Porta I/O 2
        MVI     A,0CH ; Carica la maschera nell'Accumulatore
        OUT     2     ; Uscita alla Porta I/O 2
; Alla fine del ciclo di stampa pone ad 1 il bit 1 della Porta I/O 2
; Questo pone alto CH RDY
        MVI     A,3
        OUT     3
; Ri-immagazzina i contenuti dei registri e lo stato del programma interrotto
        POP     H     ; Ri-immagazzina H ed L
        POP     D     ; Ri-immagazzina D ed E
        POP     B     ; Ri-immagazzina B e C
        POP     PSW   ; Ri-immagazzina l'Accumulatore e lo stato
        RET
```

Il modo in cui i dati vengono conservati nello stack è abbastanza diretto:



Si ricordi che per ri-immagazzinare i contenuti dei registri e dello stato nell'ordine inverso di introduzione non si avranno problemi.

GIUSTIFICAZIONE DEGLI INTERRUPTS

I programmatori di microcalcolatori e quelli di grossi calcolatori fanno un uso indiscriminato di interrupts semplicemente per dividere il costo dell'Unità di Elaborazione Centrale tra un certo numero di applicazioni diverse.

ECONOMIA DELL'INTERRUPT

Come utente di un microcalcolatore si dovrà giustificare la divisione di un costo che può essere nel range da 5 a 20 dollari. Tra questo costo si deve caricare il costo della logica esterna richiesta per originare i segnali di richiesta di interrupt e le istruzioni di restart — così come l'ulteriore costo della programmazione. **Lo scambio economico non rende certamente ovvia la percorribilità di interrupts all'interno di sistemi a microcalcolatore.** Si devono esaminare le applicazioni con cura prima di assumere che gli interrupts costituiscono la via percorribile. Una seconda CPU oppure un intero secondo sistema microcalcolatore sarà frequentemente meno costoso dell'impiego di interrupts per dividere un singolo sistema a microcalcolatore tra un certo numero di applicazioni diverse.

CONSIDERAZIONI DI TIMING DI INTERRUPT

Assumendo che gli interrupts sembrino convenienti per un'applicazione specifica, sono importanti anche le considerazioni di timing. Certamente gli interrupts sembrano molto attrattivi quando l'applicazione che si considera manipola eventi asincroni. In questo caso si **supponga che il ciclo di stampa medio duri approssimativamente 10 millisecondi; inoltre si supponga che sia possibile dire se l'intervallo di tempo tra cicli di stampa sarà 1 millisecondo o 100 millisecondi. In queste circostanze, allo scopo di eseguire qualche altro programma tra cicli di stampa, si devono usare gli interrupts per iniziare il ciclo di stampa** — poichè non si conosce quando il ciclo di stampa successivo inizierà.

In realtà il tempo che trascorre tra cicli di stampa sarà noto con grande precisione. **Una stampante avrà una velocità di stampa di carattere ben nota.** Se questa velocità è 45 caratteri al secondo allora per stampare un carattere saranno richiesti 22,2 millisecondi. Se per eseguire la routine del ciclo di stampa effettivo sono necessari 10 dei 22 millisecondi allora tra i cicli di stampa rimarranno 12 millisecondi. **Non si è lontani dal richiedere interrupt.** Mentre il programma eseguito tra cicli di stampa è interrotto in segmenti, ognuno dei quali è eseguito in 12 millisecondi o meno, quindi ogni segmento può terminare con un anello di istruzione che prova lo stato dell'ingresso di decodifica di velocità allo scopo di iniziare il ciclo di stampa successivo:

```
LOOP IN     2      ; Ingresso ai contenuti della Porta I/O 2
     ANI    20H    ; Isola il segnale di decodifica di velocità
     JNZ    LOOP   ; Se ancora 1 il nuovo ciclo di stampa non deve cominciare
```

PERDITA DI TEMPO DEL SERVIZIO INTERRUPT

Esiste una penalità di tempo associata con ogni interrupt che viene elaborato. Si osservino le istruzioni che devono essere eseguite prima e dopo il programma del ciclo di stampa stesso; ci sono quattro istruzioni Push, 4 istruzioni Pop, una istruzione Call ed una istruzione Return. Si sommino i numeri di cicli richiesti e si troverà che sono richiesti 111 cicli per eseguire queste istruzioni — e questo significa **55,5 microsecondi per interrupt.**

Questo è approssimativamente il 5% del tempo totale richiesto per eseguire l'intero programma del ciclo di stampa.

Si proietti questo tempo aggiuntivo in un sistema più complesso dove per esempio 10 sorgenti di logica esterna possono generare una richiesta di interrupt. Tale schema di interrupt complesso può sembrare ragionevole per un programmatore di microcalcolatori ma all'interno del sistema a microcalcolatore è possibile che per 55,5 microsecondi impiegati non produttivamente come tempo aggiuntivo, il 50% del tempo il sistema a microcalcoaltore non può fare niente di più di conservare e ri-immagazzinare registri e stato. Chiaramente ci si deve avvicinare agli interrupt con qualche cautela.

INTERRUPT MULTIPLI

Si supponga che l'applicazione che si considera sia tale che non si possono applicare tutti gli avvertimenti contro l'impiego di interrupt. Si può, per esempio, avere un'applicazione in cui un certo numero di eventi asincroni si verifica abbastanza di rado in modo che essi non influiscono seriamente sul tempo di esecuzione disponibile. **Ci sono innumerevoli modi di realizzare interrupt multipli in un sistema a microcalcolatore tipo 8080** ed è certamente lontano dallo scopo del libro esplorarli tutti. Ci sono comunque due concetti base di tutti gli schemi di interrupt multiplo disponibili:

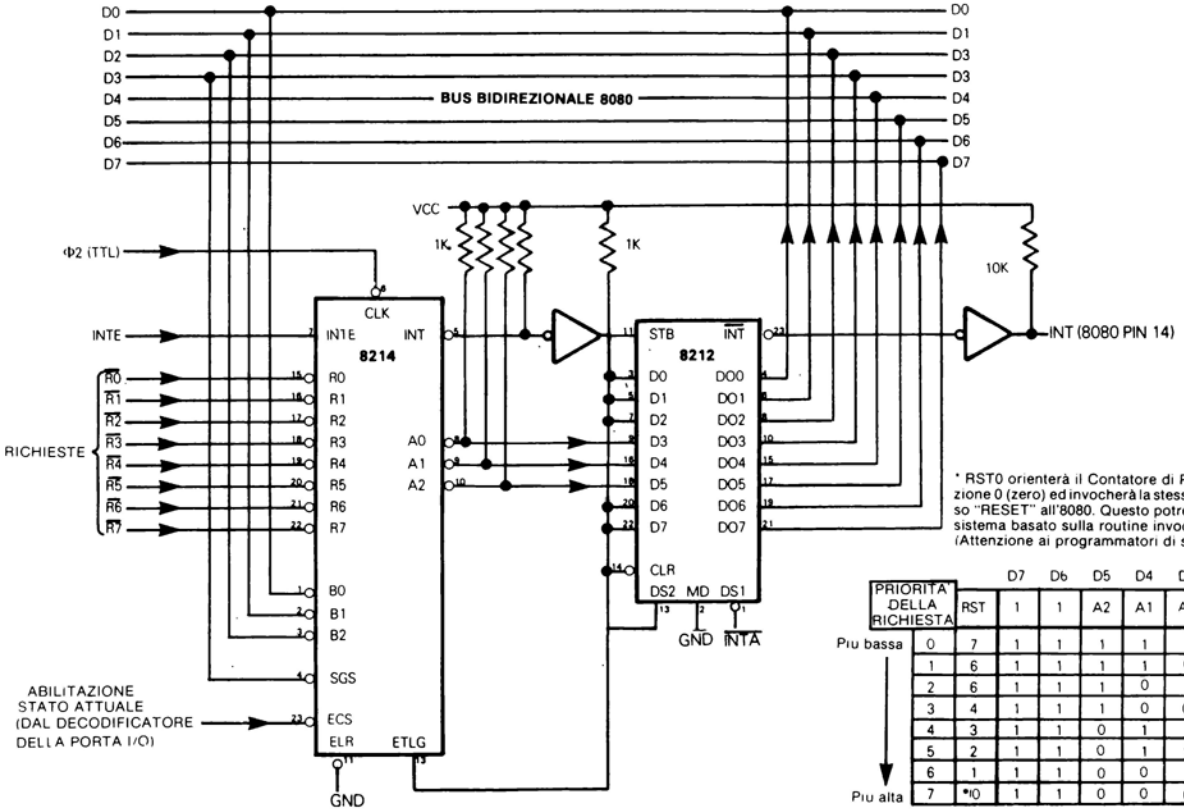
1) Invece del collegamento diretto della richiesta di interrupt esterna col pin richiesta interrupt della CPU, viene impiegato l'8255 PPI o l'8212 porta I/O per fare da buffer e trasmettere le richieste di interrupt.

2) Fornendo otto o meno interrupt esterni da elaborare le istruzioni Restart opportune descriminano tra gli interrupt. Una Porta I/O 8255 può essere impiegata con un decodificatore uno ad otto per generare l'appropriata istruzione Restart come segue:

Se si hanno otto interrupt esterni si ricordi che uno di questi impiegherà Restart RST0 — e questo coinciderà con la condizione di Reset poichè entrambe causano la esecuzione del programma dirottata alla locazione di memoria 0.

Se si hanno sette o meno interrupt esterni diversi, la loro elaborazione è relativamente diretta.

Se si hanno note o più interrupt esterni e si sta cercando di elaborarli impiegando una CPU, con tutta probabilità c'è qualche errore nel modo in cui il sistema a microcalcolatore è stato progettato. E' possibile immaginare un'applicazione nella quale un complesso schema di interrupt è economicamente realizzato attraverso un sistema a microcalcolatore con una CPU. Questo non è il nostro caso.



* RST0 orienterà il Contatore di Programma alla locazione 0 (zero) ed invocherà la stessa routine dell'ingresso "RESET" all'8080. Questo potrebbe ri-inizializzare il sistema basato sulla routine invocata. (Attenzione ai programmatori di sistema)

PRIORITA' DELLA RICHIESTA		RST	D7	D6	D5	D4	D3	D2	D1	D0
			1	1	A2	A1	A0	1	1	1
Piu bassa	0	7	1	1	1	1	1	1	1	1
	1	6	1	1	1	1	0	1	1	1
	2	6	1	1	1	0	1	1	1	1
	3	4	1	1	1	0	0	1	1	1
	4	3	1	1	0	1	1	1	1	1
	5	2	1	1	0	1	0	1	1	1
	6	1	1	1	0	0	1	1	1	1
Piu alta	7	*0	1	1	0	0	0	1	1	1

Figura 5-2. Generazione del codice dell'istruzione restart seguente un interrupt, per un sistema a microcalcolatore 8080

Capitolo 6

SET DI ISTRUZIONI PER L'8080/9080

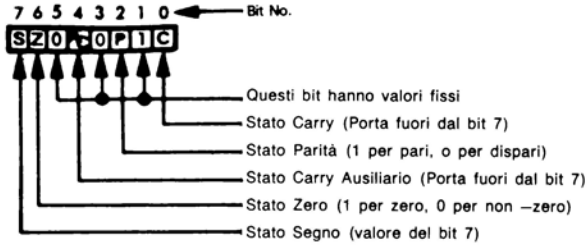
ABBREVIAZIONI

Queste sono le abbreviazioni usate in questo capitolo:

A	L'Accumulatore
B	Il registro B } Questi sono a volte considerati come una coppia di registri
C	Il registro C
D	Il registro D } Questi sono a volte considerati come una coppia di registri
E	Il registro E
H	Il registro H } Questa coppia di registri fornisce l'indirizzamento implicato della memoria
L	Il registro L
CS	Indicatore Carry
A_C	Indicatore Carry Ausiliario
ZS	Indicatore Zero
SS	Indicatore Segno
PS	Indicatore Parità
I	Il registro di istruzione
I2	Secondo byte codice oggetto
I3	Terzo byte codice oggetto
PC	Il Contatore di Programma
SP	Il Puntatore dello Stack
PSW	Il Program Status Word che ha bit assegnati ai flag indicatori come mostrato alla pagina successiva
H	Comparendo alla fine di un gruppo di digit (per esempio 213 AH) sta a designare i digit esadecimali
dati	Dati ad 8 bit immediati
dev	Un Dispositivo I/O
dati 16	Dati a 16 bit immediati
reg	Registri A, B, C, D, E, H od L
M	Memoria, indirizzo implicato da HL
label	Un indirizzo a 16 bit, specificando una label di istruzione
rp	Una coppia di registri: B per BC, D per DE, H per HL, SP per Puntatore dello Stack, PSW per flag indicatori ed Accumulatore
port	Una porta I/O, identificata da un numero tra 0 ed FF_{16}
addr	Un indirizzo a 16 bit, specificando un byte della memoria dati
[ ]	Contenuti della locazione identificata dentro le parentesi
[[ ]]	Parola della memoria indirizzata dalla locazione identificata dentro la parentesi
←	Muovi i dati in direzione della freccia
←→	Scambia i contenuti della locazioni ad entrambi i lati della freccia
+	Somma
—	Sottrai
., ∧	AND
V	OR
⊕, ⊻	XOR

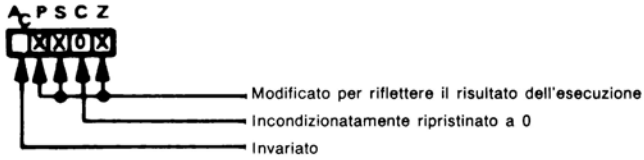
STATO

I cinque flag di stato sono memorizzati in un registro Program Status Word (PSW) come segue:



PSW ed A sono talvolta trattati come una coppia di registri.

L'effetto dell'assegnazione di un'istruzione sullo stato è illustrato come segue:



CAMBIAMENTI DELLO STATO CON L'ESECUZIONE DI ISTRUZIONI

Nelle illustrazioni di esecuzione di istruzione, una X identifica uno stato imposto o ripristinato. Uno 0 identifica uno stato che è sempre basso. Un bianco significa che lo stato non cambia.

MNEMONICI DI ISTRUZIONE

La parte fissa di una istruzione in linguaggio assembly è mostrata nel CASO in ALTO.

La parte variabile (dati immediati, numero disposti I/O, nome del registro, label o indirizzo) sono mostrati nel caso in basso.

CODICI OGGETTO DI ISTRUZIONE

I codici oggetto di istruzione sono rappresentati con due esadecimali per istruzioni senza variazioni.

I codici oggetto di istruzione sono rappresentati con otto digit binari per istruzioni con variazioni; e così identificabile la rappresentazione in digit binari di variazioni.

TEMPO DI ESECUZIONE DI ISTRUZIONE E CODICI

La Tabella 6-2 elenca le istruzioni in ordine alfabetico, mostrando i codici oggetto ed i tempi di esecuzione espressi in cicli macchina.

Dove sono mostrati due cicli di istruzione, il primo è per "condizione non convergente", mentre il secondo è per "condizione convergente".

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
I/O	IN	DEV	2						[A] ← [DEV] Ingresso da A al dispositivo DEV (DEV da 0 a 255)
	OUT	DEV	2						[DEV] ← [A] Uscita da A al dispositivo DEV (DEV da 0 a 255)
Primaria Reference di Memoria	LDAX	RP	1						[A] ← [[RP]] Carica A usando l'indirizzo implicato da BC (RP = B) o DE (RP = D)
	STAX	RP	1						[[RP]] ← [A] Memorizza A usando l'indirizzamento implicato come per LDAX
	MOV	R,M	1						[R] ← [[HL]] Carica tutti i registri usando l'indirizzo implicato da HL
	MOV	M,R	1						[[HL]] ← [R] Memorizza tutti i registri usando l'indirizzo implicato da HL
	LDA	ADDR	3						[A] ← [ADDR], cioè [A] ← [[I3,I2]] Carica A, usa l'indirizzamento diretto
	STA	ADDR	3						[ADDR] ← [A], cioè [[I3,I2]] ← [A] Carica A, usa l'indirizzamento diretto
	LHLD	ADDR	3						[L] ← [ADDR], [H] ← [ADDR + 1] cioè, [L] ← [[I3,I2]], [H] ← [[I3,I2] + 1] Carica i registri H ed L, usa l'indirizzamento diretto
	SHLD	ADDR	3						[ADDR] ← [L], [ADDR + 1] ← [H] cioè, [[I3,I2]] ← [L], [[I3,I2] + 1] ← [H] Memorizza i registri H ed L, usa l'indirizzamento diretto

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Secondaria Reference di Memoria Opera Memoria	ADD	M	1	X	X	X	X	X	[A] ← [A] + [[HL]] Somma ad A
	ADC	M	1	X	X	X	X	X	[A] ← [A] + [[HL]] + [CS] Somma ad A con Carry
	SUB	M	1	X	X	X	X	X	[A] ← [A] - [[HL]] Sottrai da A
	SBB	M	1	X	X	X	X	X	[A] ← [A] - [[HL]] - [CS] Sottrai da A con prestito
	ANA	M	1	0	X**	X	X	X	[A] ← [A] ∧ [[HL]] AND con A
	XRA	M	1	0	0	X	X	X	[A] ← [A] ⊻ [[HL]] OR – Esclusivo con A
	ORA	M	1	0	0	X	X	X	[A] ← [A] ∨ [[HL]] OR con A
	CMP	M	1	X	X	X	X	X	[A] - [[HL]] Paragona con A
	INR	M	1		X	X	X	X	[[HL]] ← [[HL]] + 1 Incrementa Memoria
	DCR	M	1		X	X	X	X	[[HL]] ← [[HL]] - 1 Decrementa Memoria
Immediato	LXI	RP,DATA16	3						[RP] ← DATA16 Carica dati a 16-bit immediati in BC (RP = B), DE (RP = D), HL (RP = H) o SP (RP = SP)
	MVI	M,DATA	2						[[HL]] ← DATA Carica dati ad 8-bit immediato nella locazione di memoria avente indirizzo implicato da HL
	MVI	R,DATA	2						[R] ← DATA Carica dati ad 8-bit immediato in qualsiasi registro

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Salto	JMP	ADDR	3						[PC] ← ADDR Salta all'istruzione con la label ADDR
	PCHL		1						[PC] ← [HL] Salta all'istruzione con indirizzo contenuto in HL
Chiamata e Ritorno da Subroutine (Immediato e Stack)	CALL	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine iniziante ad ADDR
	CC	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se C = 1
	CNC	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se C = 0
	CZ	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se Z = 1
	CNZ	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se Z = 0
	CP	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se S = 0
	CM	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se S = 1
	CPE	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se parità pari
	CPO	ADDR	3						[[SP]] ← [PC], [PC] ← ADDR, [SP] ← [SP] - 2 Salta alla subroutine se parità dispari
	RET		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Chiamata e Ritorno da Subroutine (Immediato e Stack)	RC		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se C = 1
	RNC		1*						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se C = 0
	RZ		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se Z = 1
	RNZ		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se Z = 0
	RM		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se S = 1
	RP		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se S = 0
	RPE		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se parità pari
	RPO		1						[PC] ← [[SP]], [SP] ← [SP] + 2 Ritorna dalla subroutine se parità dispari
Opera Immediato	ADI	DATA	2	X	X	X	X	X	[A] ← [A] + DATA Somma immediato ad A
	ACI	DATA	2	X	X	X	X	X	[A] ← [A] + DATA + [CS] Somma con carry immediato ad A
	SUI	DATA	2	X	X	X	X	X	[A] ← [A] - DATA Sottrai immediato da A
	SBI	DATA	2	X	X	X	X	X	[A] ← [A] - DATA - [CS] Sottrai immediato con prestito da A
	ANI	DATA	2	0	X**	X	X	X	[A] ← [A] ∧ DATA AND immediato con A

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Opera Immediato	XRI	DATA	2	0	0	X	X	X	[A] ← [A] ⊻ DATA OR – esclusivo immediato con A
	ORI	DATA	2	0	0	X	X	X	[A] ← [A] V DATA OR immediato con A
	CPI	DATA	2	X	X	X	X	X	[A] - DATA Confronta immediato con A
Salto Condizionato	JC	ADDR	3						[PC] ← ADDR Salta se C = 1
	JNC	ADDR	3						[PC] ← ADDR Salta se C = 0
	JZ	ADDR	3						[PC] ← ADDR Salta se Z = 1
	JNZ	ADDR	3						[PC] ← ADDR Salta se Z = 0
	JP	ADDR	3						[PC] ← ADDR Salta se S = 0
	JM	ADDR	3						[PC] ← ADDR Salta se S = 1
	JPE	ADDR	3						[PC] ← ADDR Salta con parità pari
	JPO	ADDR	3						[PC] ← ADDR Salta con parità dispari

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Movimento Reg – Reg	MOV	D,S	1						[R] ← [R] Muovi qualsiasi registro (S) in qualunque altro (D)
	XCHG		1						[D] ←→ [H], [E] ←→ [L] Scambia DE con HL
	SPHL		1						[HL] → [SP] Manda HL in SP
Operazioni Registro-Registro	ADD	R	1	X	X	X	X	X	[A] ← [A] + [R] Somma qualunque registro ad A
	ADC	R	1	X	X	X	X	X	[A] ← [A] + [R] + [CS] Somma con Carry qualsiasi registro ad A
	SUB	R	1	X	X	X	X	X	[A] ← [A] - [R] Sottrai qualsiasi registro da A
	SBB	R	1	X	X	X	X	X	[A] ← [A] - [R] - [CS] Sottrai qualsiasi registro con prestito da A
	ANA	R	1	0	X**	X	X	X	[A] ← [A] ∧ [R] AND di qualunque registro con A
	XRA	R	1	0	0	X	X	X	[A] ← [A] ⊻ [R] OR – esclusivo di qualunque registro con A
	ORA	R	1	0	0	X	X	X	[A] ← [A] ∨ [R] OR qualunque registro con A
	CMP	R	1	X	X	X	X	X	[A] - [R] Confronta qualunque registro con A
Opera Registro	INR	R	1		X	X	X	X	[R] ← [R] + 1 Incrementa qualsiasi registro
	DCR	R	1		X	X	X	X	[R] ← [R] - 1 Decrementa qualsiasi registro
	CMA		1						[A] ← [$\overline{A}$] Complementa A

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Opera Registro	DAA		1	X	X	X	X	X	Aggiusta i decimali di A
	RLC		1	X					Ruota A a sinistra con ramificazione del carry
	RRC		1	X					Ruota A a destra con ramificazione del carry
	RAL		1	X					Ruota A a sinistra con carry
	RAR		1	X					Ruota A a destra con carry
	DAD	RP	1	X					[HL] ← [HL] + [RP] Somma RP ad HL. RP = BC, DE, HL o SP
	INX	RP	1						[RP] ← [RP] + 1 Incrementa RP. RP = BC, DE, HL o SP
	DCX	RP	1						[RP] ← [RP] - 1 Decrementa RP. RP = BC, DE, HL o SP
Stack	PUSH	RP	1						[[SP]] ← [RP], [SP] ← [SP] - 2 Carica il contenuto di RP nello Stack
	POP	RP	1						[RP] ← [[SP]], [SP] ← [SP] + 2 Preleva lo stack in RP RP=BC, DE, HL o SP
	XTHL		1						[HL] ⟷ [[SP]] Scambia HL con sommità dello Stack

Tabella 6-1. Sommario del Set di Istruzioni del Microcalcolatore 8080/9080
(Continua)

Tipo	Mnemonico	Operando/i	Byte	Stato					Operazione Eseguita
				C	A_C	Z	S	P	
Interrupt	EI		1						Abilita interrupt dopo l'esecuzione della prossima istruzione
	RIM		1						Leggi Interrupt Mask*
	DI		1						Disabilita Interrupt
	SIM		1						Poni Interrupt Mask*
	RST		1						Riparti
Stato	STC		1	1					[CS] ← 1 Poni Carry
	CMC		1	X					[CS] ← [CS] Complementa Carry
	NOP		1						
	HLT		1						

Stati:
C = Carry
A_C = Carry oltre il 3 bit
Z = Zero
S = Segno
P = Parità
X = Stato set o reset
0 = Stato reset

Bianco = Stato invariato
*Istruzioni 8085
**8085 pone A_C ad 1 per tutte le istruzioni AND

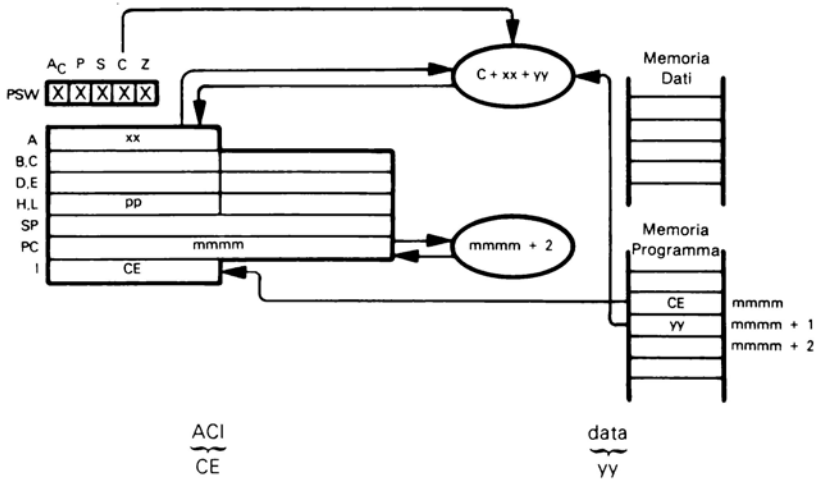
Tabella 6-2. Sommario dei Codici Oggetto delle Istruzioni e dei Cicli di Esecuzione

Istruzione		Cod. Oggetto	Byte	Cycles	Istruzione		Cod. Oggetto	Byte	Cicli
ACI	DATA	CE yy	2	7	LXI	RP,DATA16	00xx0001	3	10
ADC	REG	10001xxx	1	4			yyyy		
ADC	M	8E	1	7	MOV	REG,REG	01dddsss	1	5
ADD	REG	10000xxx	1	4	MOV	M,REG	01110sss	1	7
ADD	M	86	1	7	MOV	REG,M	01ddd110	1	7
ADI	DATA	C6 yy	2	7	MVI	REG,DATA	00ddd110	2	7
ANA	REG	10100xxx	1	4			yy		
ANA	M	A6	1	7	MVI	M,DATA	36 yy	2	10
ANI	DATA	E6 yy	2	7	NOP		00	1	4
CALL	LABEL	CD ppqq	3	17	ORA	REG	10110xxx	1	4
CC	LABEL	DC ppqq	3	11/17	ORA	M	B6	1	7
CM	LABEL	FC ppqq	3	11/17	ORI	DATA	F6 yy	2	7
CMA		2F	1	4	OUT	PORT	D3 yy	2	10
CMC		3F	1	4	PCHL		E9	1	5
CMP	REG	10111xxx	1	4	POP	RP	11xx0001	1	10
CMP	M	BE	1	7	PUSH	RP	11xx0101	1	11
CNC	LABEL	D4 ppqq	3	11/17	RAL		17	1	4
CNZ	LABEL	C4 ppqq	3	11/17	RAR		1F	1	4
CP	LABEL	F4 ppqq	3	11/17	RC		D8	1	5/11
CPE	LABEL	EC ppqq	3	11/17	RET		C9	1	10
CPI	DATA	FE yy	2	7	RIM		20	1	4*
CPO	LABEL	E4 ppqq	3	11/17	RLC		07	1	4
CZ	LABEL	CC ppqq	3	11/17	RM		F8	1	5/11
DAA		27	1	4	RNC		D0	1	5/11
DAD	RP	00xx1001	1	10	RNZ		C0	1	5/11
DCR	REG	00xxx101	1	5	RP		F0	1	5/11
DCR	M	35	1	10	RPE		E8	1	5/11
DCX	RP	00xx1011	1	5	RPO		E0	1	5/11
DI		F3	1	4	RRC		0F	1	4
EI		FB	1	4	RST	N	11xxx111	1	11
HLT		76	1	7	RZ		C8	1	5/11
IN	PORT	DB yy	2	10	SBB	REG	10011xxx	1	4
INR	REG	00xxx100	1	5	SBB	M	9E	1	7
INR	M	34	1	10	SBI	DATA	DE yy	2	7
INX	RP	00xx0011	1	5	SHLD	ADDR	22 ppqq	3	16
JC	LABEL	DA ppqq	3	10	SIM		30	1	4*
JM	LABEL	FA ppqq	3	10	SPHL		F9	1	5
JMP	LABEL	C3 ppqq	3	10	STA	ADDR	32 ppqq	3	13
JNC	LABEL	D2 ppqq	3	10	STAX	RP	000x0010	1	7
JNZ	LABEL	C2 ppqq	3	10	STC		37	1	4
JP	LABEL	F2 ppqq	3	10	SUB	REG	10010xxx	1	4
JPE	LABEL	EA ppqq	3	10	SUB	M	96	1	7
JPO	LABEL	E2 ppqq	3	10	SUI	DATA	D6 yy	2	7
JZ	LABEL	CA ppqq	3	10	XCHG		EB	1	4
LDA	ADDR	3A ppqq	3	13	XRA	REG	10101xxx	1	4
LDAX	RP	000x1010	1	7	XRA	M	AE	1	7
LHLD	ADDR	2A ppqq	3	16	XRI	DATA	EE yy	2	7
					XTHL		E3	1	18

ppqq	rappresenta quattro indirizzi di memoria in digit esadecimali
yy	rappresenta due digit di dati esadecimali
yyyy	rappresenta quattro digit di dati esadecimali
x	rappresenta un digit binario opzionale
ddd	rappresenta i digit binari opzionali identificanti un registro di destinazione
sss	rappresenta i digit binari opzionali identificanti un registro sorgente

*Istruzioni 8085

ACI — SOMMA CON CARRY IMMEDIATO ALL'ACCUMULATORE



Somma il contenuto del successivo byte della memoria di programma e lo stato Carry all'Accumulatore.

Si supponga xx = $3A_{16}$, yy = $7C_{16}$, C = 0. Dopo che l'istruzione

ACI 7CH

è stata eseguita, l'Accumulatore conterrà B6:

```
3A    =  00111010
7C    =  01111100
Carry =         0
         10110110
```

Non c'è carry pone C = 0

1 pone S a 1

Cinque bit 1, poni P a 0

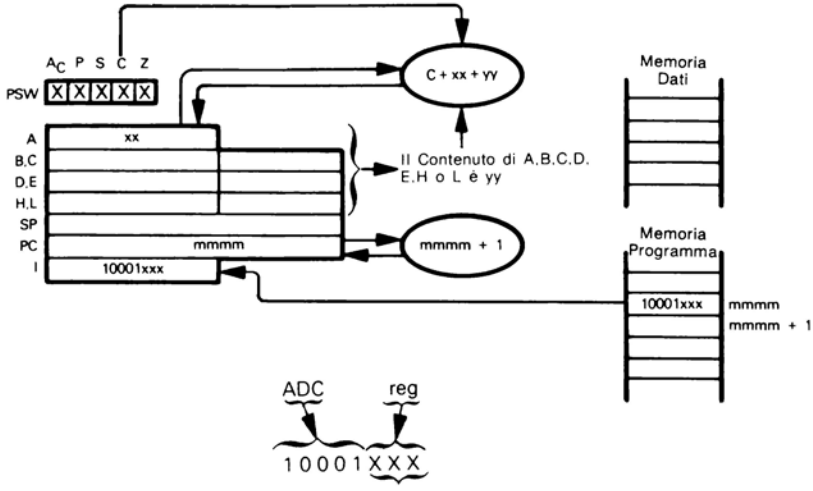
Risulta Non-zero, poni Z a 0

Carry pone Ac a 1

Questa è una istruzione di manipolazione dati di routine.

ADC — SOMMA DI UN REGISTRO O MEMORIA CON CARRY NELL'ACCUMULATORE

Questa istruzione assume due forme. La prima considera i contenuti di un registro sommati all'Accumulatore:



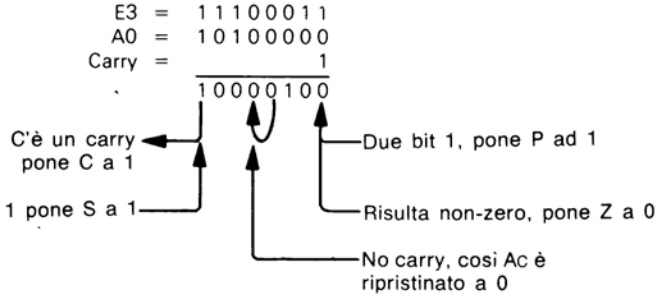
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Somma il contenuto del registro A, B, C, D, E, H o L e lo stato Carry all'Accumulatore.

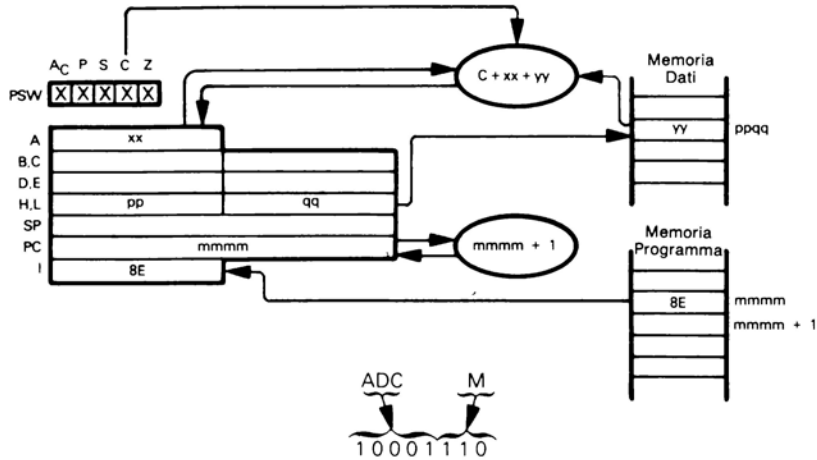
Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$, C = 1. Dopo che l'istruzione:

ADC E

e stata eseguita l'Accumulatore conterrà 84_{16}:



I contenuti del byte di memoria possono anche essere sommati, con Carry, all'Accumulatore:



Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1, allora l'esecuzione dell'istruzione:

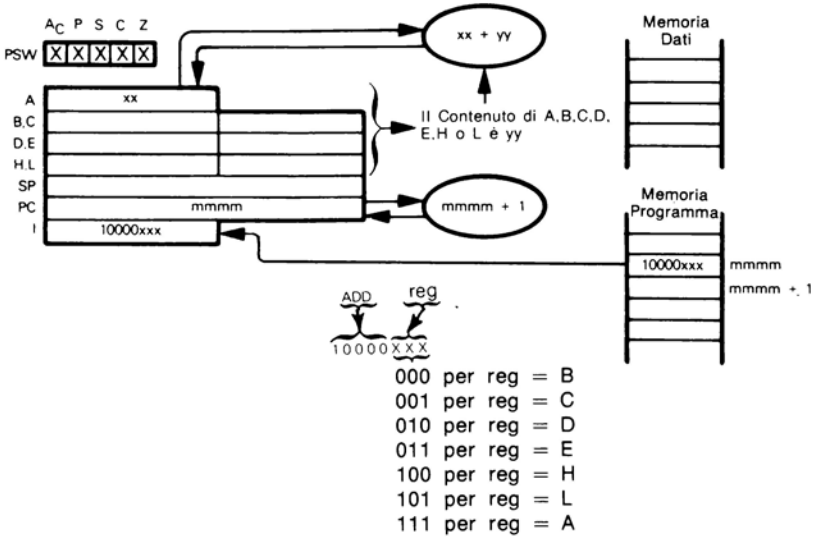
ADC M

genera lo stesso risultato dell'esecuzione dell'istruzione ADC E appena descritta.

L'istruzione ADC è principalmente usata nell'addizione multibyte, per il secondo e successivi byte.

ADD — SOMMA DI REGISTRO O MEMORIA ALL'ACCUMULATORE

Questa istruzione assume due forme. La prima considera i contenuti di un registro sommati all'Accumulatore:

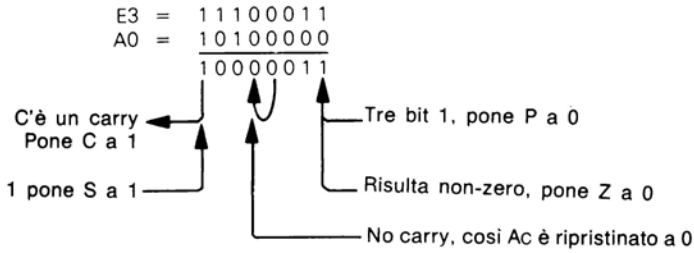


Somma i contenuti del registro A, B, C, D, E, H o L all'Accumulatore.

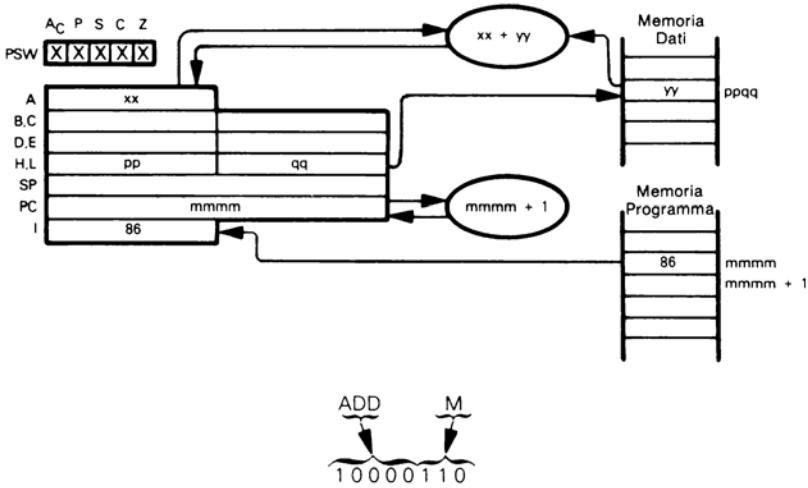
Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$, C = 1. Dopo che l'istruzione:

ADD E

è stata eseguita, l'Accumulatore conterrà 83_{16}:



I contenuti del byte di memoria possono anche essere sommati all'Accumulatore:



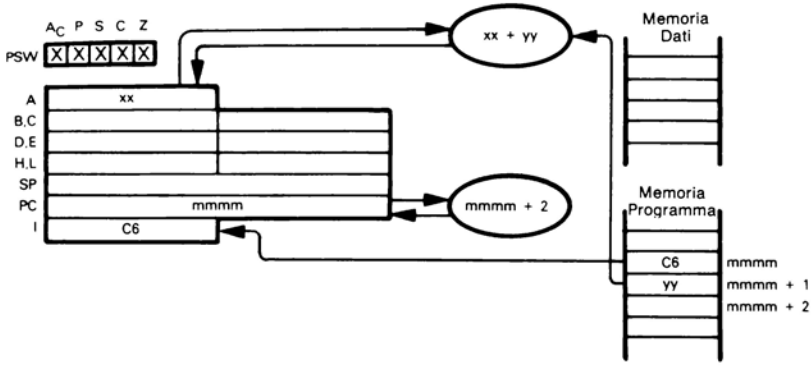
Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1 allora l'esecuzione dell'istruzione:

ADD M

genera lo stesso risultato dèll'esecuzione dell'istruzione ADD E appena descritta.

ADD è l'istruzione dell'addizione binaria usata nelle operazioni normali, a singolo byte; è anche l'istruzione usata per sommare i byte di basso ordine di due numeri multibyte.

ADI – SOMMA IMMEDIATA ALL'ACCUMULATORE



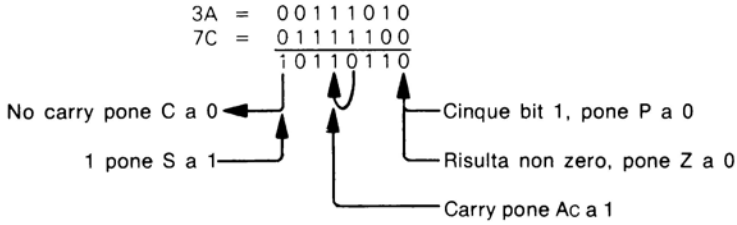
ADI data
C6 yy

Somma il contenuto del successivo byte della memoria di programma all'Accumulatore.

Si supponga xx = $3A_{16}$, yy = $7C_{16}$, C = 0. Dopo che l'istruzione:

ADI 7CH

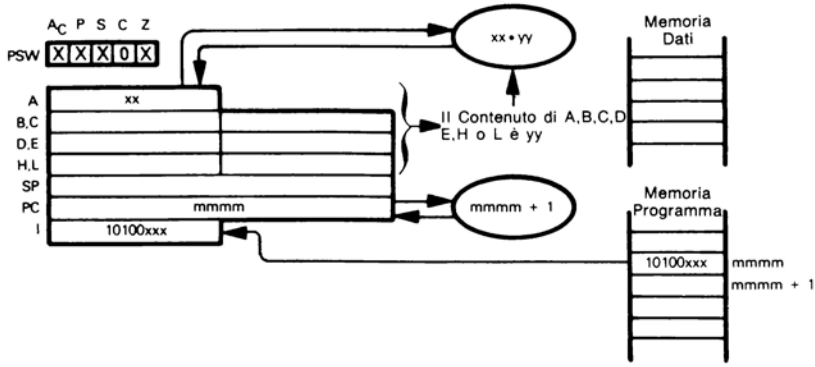
è stata eseguita l'Accumulatore conterrà B6:

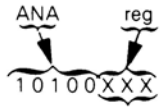


Questa è una istruzione di manipolazione dati di routine.

ANA — AND DI REGISTRO O MEMORIA CON L'ACCUMULATORE

Questa istruzione assume due forme. La prima esegue l'AND fra contenuto del registro e l'Accumulatore:





000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Opera l'AND tra l'Accumulatore ed il registro A, B, C, D, E, H o L ed il risultato resta nell'Accumulatore.

Si supponga xx = $E3_{16}$, il registro E contenga $A0_{16}$. Dopo che l'istruzione:

ANA E

è stata eseguita, l'Accumulatore conterrà $A0_{16}$:

```
E3 =  11100011
A0 =  10100000
      --------
      10100000
```

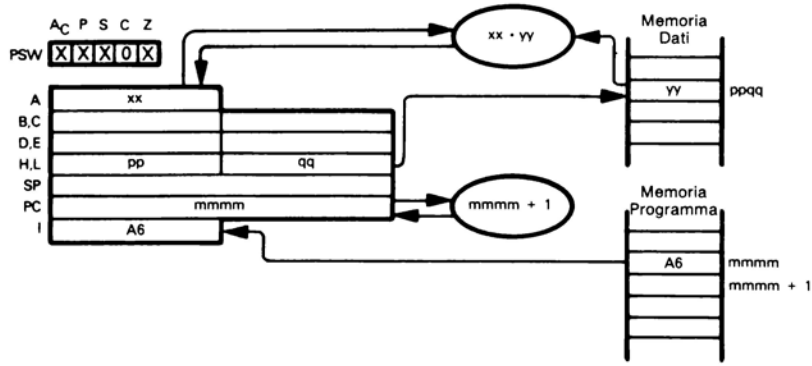
Il carry è sempre posto a 0

1 pone S ad 1

Due byt 1, pone P ad 1

Risulta non zero, pone Z a 0

Si può operare l'AND tra i contenuti di un byte di memoria e l'Accumulatore:

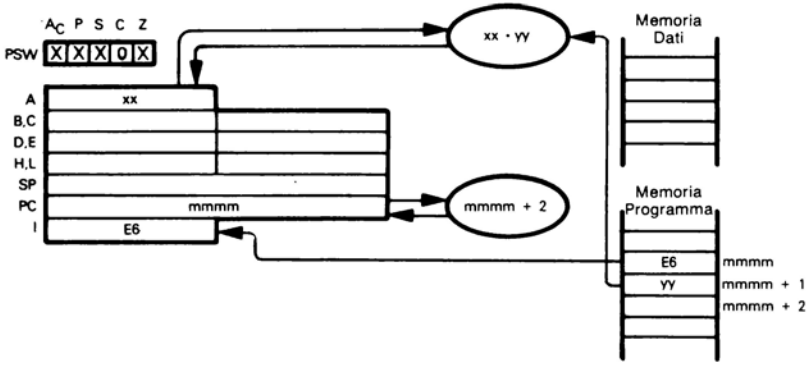


Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1, allora l'esecuzione dell'istruzione:

ANA M

genera lo stesso risultato dell'esecuzione dell'istruzione ANA E, appena descritta. ANA è una istruzione logica usata frequentemente.

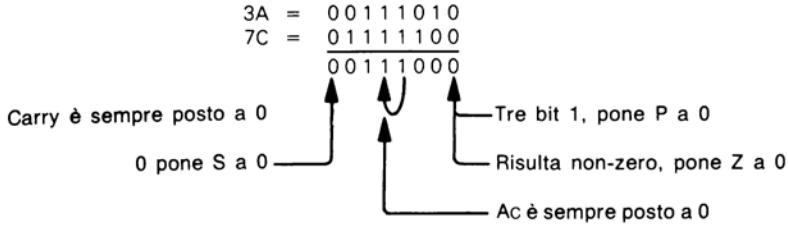
ANI – AND IMMEDIATO CON L'ACCUMULATORE



Opera l'AND fra il contenuto del successivo byte della memoria di programma e l'Accumulatore. Si supponga XX = $3A_{16}$, yy = $7C_{16}$. Dopo che l'istruzione:

ANI 7CH

è stata eseguita, l'Accumulatore conterrà 38_{16}.

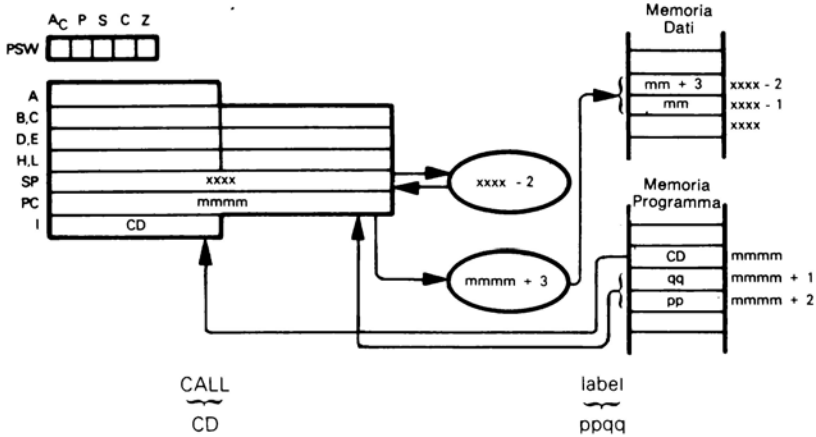


Questa è una istruzione logica di routine; è usata spesso per porre "off" dei bit. Per esempio l'istruzione:

ANI 7FH

porrà incondizionatamente a 0 il bit di ordine elevato dell'Accumulatore.

CALL – CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO



Memorizza l'indirizzo dell'istruzione seguente CALL alla sommità dello stack; la sommità dello stack è un byte della memoria dati, indirizzato dal Puntatore dello Stack. Poi sottrae 2 dal Puntatore dello Stack in modo da indirizzare la nuova sommità dello Stack. Muove l'indirizzo a 16 bit contenuto nel secondo e terzo byte dell'istruzione CALL del programma oggetto e lo riporta nel Contatore di Programma.

Si consideri la sequenza di istruzioni:

```
        CALL    SUBR
        ANI     7CH
        .
        .
        .
SUBR
```

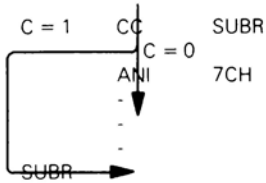
Dopo che l'istruzione CALL è stata eseguita l'indirizzo dell'istruzione ANI è contenuto alla sommità dello stack. Il Puntatore dello Stack è decrementato di 2. Poi verrà eseguita l'istruzione con la label SUBR.

CC — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO CARRY E' UGUALE AD 1

CC
DC

Questa istruzione è identica alla istruzione CALL tranne il fatto che la subroutine identificata sarà chiamata solo se lo stato CARRY è uguale a 1, diversamente sarà eseguita la istruzione sequenzialmente seguente l'istruzione CC.

Si consideri la seguente sequenza di istruzioni:



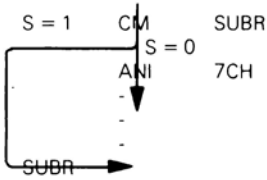
Dopo che l'istruzione CC è stata eseguita, se lo stato Carry non è uguale ad 1, verrà eseguita l'istruzione ANI. Se lo stato Carry è uguale ad 1 l'indirizzo della istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. L'istruzione con la label SUBR sarà la prossima ad essere eseguita.

CM — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN E' UGUALE AD 1

CM
FC

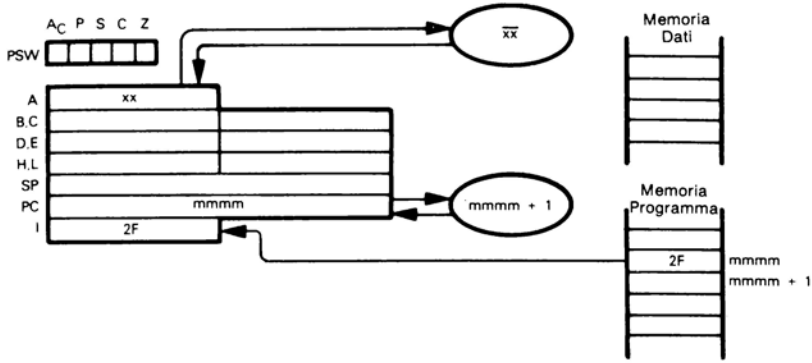
Questa istruzione è identica alla istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato SIGN è uguale a 1; diversamente sarà eseguita la istruzione sequenzialmente seguente l'istruzione CM.

Si consideri la sequenza di istruzioni:



Dopo che è stata eseguita l'istruzione CM, se lo stato Sign non è uguale ad 1, verrà eseguita l'istruzione ANI. Se lo stato Sign è uguale ad 1, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CMA — COMPLETA L'ACCUMULATORE



CMA
2F

Complementa il contenuto dell'Accumulatore. Non viene influenzato il contenuto di nessun altro registro o stato.

Si suppone che l'Accumulatore contenga $3A_{16}$. Dopo che l'istruzione

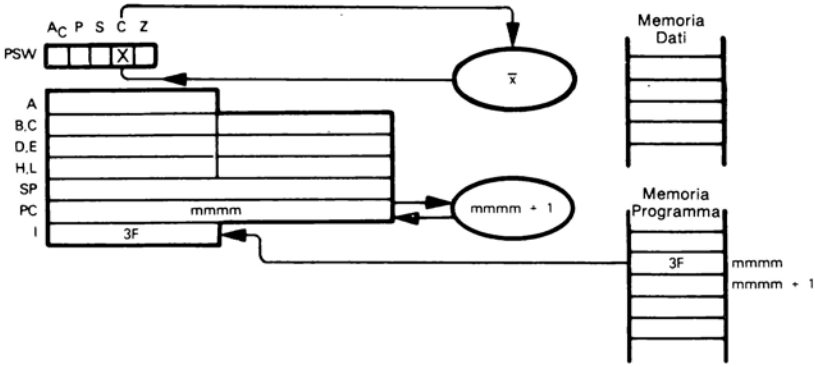
CMA

è stata eseguita, l'Accumulatore conterrà $C5_{16}$.

$3A_{16}$ = 00111010
Complemento = 11000101

Questa è una tipica istruzione logica di routine. **Non si usa per la sottrazione binaria.** Allo scopo esistono istruzioni speciali per la sottrazione (SUB ed SBB).

CMC — COMPLEMENTA LO STATO CARRY



CMC
3F

Complementa lo stato Carry. Non viene influenzato nessun altro stato o contenuto di registro.

Si supponga che lo stato Carry contenga 1. Dopo che l'istruzione

CMC

è stata eseguita lo stato Carry conterrà 0.

Questa istruzione viene usata per forzare lo stato Carry a 0 per mezzo della sequenza di istruzioni:

```
STC     , Pone lo stato Carry ad 1
CMC     ; Complementa lo stato Carry
```

Si noti che si può porre lo stato CARRY a 0 per mezzo della istruzione

ANA A

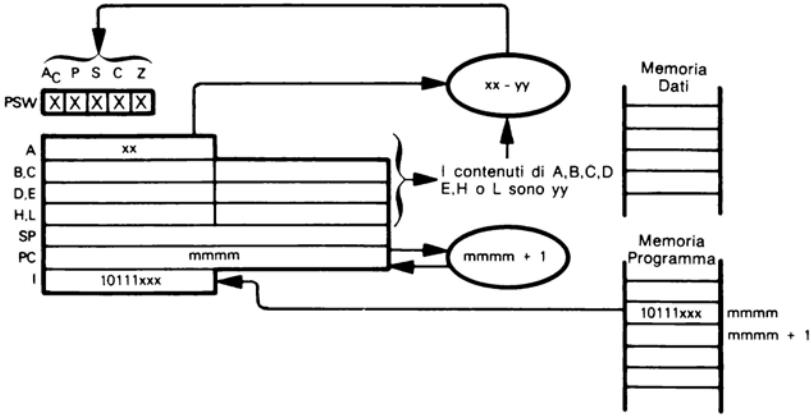
che automaticamente azzera lo stato Carry ma non modifica nessun altro contenuto di registro poichè viene l'AND dell'Accumulatore con sé stesso. L'istruzione

ORA A

serve allo stesso scopo.

CMP — CONFRONTA REGISTRO E MEMORIA CON L'ACCUMULATORE

Questa istruzione assume due forme. Prima si consideri il contenuto di un registro nei confronti dell'Accumulatore:



CMP reg

1 0 1 1 1 X X X

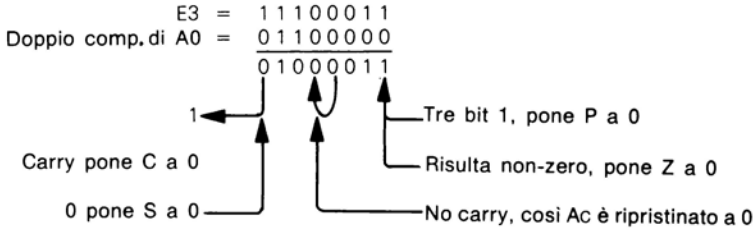
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae i contenuti di registro A, B, C, D, E, H o L dal contenuto dell'Accumulatore, considerando entrambi i numeri come semplici dati binari. Viene poi scaricato il risultato, cioè caricato il solo Accumulatore ma modificati i flag di stato per riflettere il risultato della sottrazione.

Si supponga che xx = $E3_{16}$, il registro E contenga $A0_{16}$. Dopo che l'istruzione:

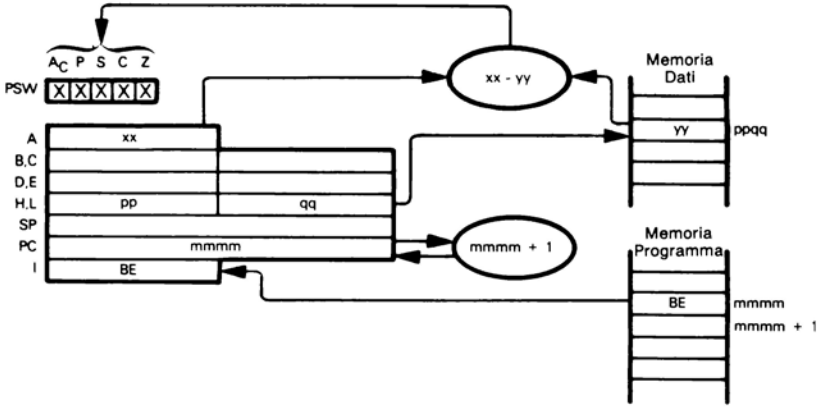
CMP E

è stata eseguita l'Accumulatore conterrà $E3_{16}$ ma gli stati saranno modificati come segue:



Si noti che il Carry risulta complementato.

Anche i contenuti di un byte di memoria possono essere paragonati con l'Accumulatore:



Se xx = $E3_{16}$ ed yy = $A0_{16}$, allora la esecuzione dell'istruzione:

CMP M

genera lo stesso risultato dell'esecuzione dell'istruzione CMP E, appena descritta.

Le istruzioni di confronto frequentemente precedono le istruzioni di Call condizionale, Ritorno e Salto.

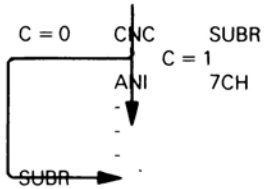
L'istruzione Paragona Immediato (CPI) è più pratica dell'istruzione CMP.

CNC — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO. MA SOLTANTO SE LO STATO CARRY E' UGUALE A 0

CNC
D4

Questa istruzione è identica all'istruzione CALL salvo che qui la istruzione identificata verrà chiamata solo se lo stato Carry è uguale a 0; diversamente verrà eseguita l'istruzione sequenzialmente seguente l'istruzione CNC.

Si consideri la sequenza di istruzioni:



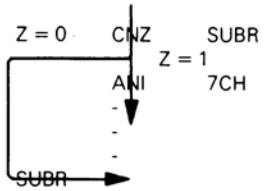
Dopo che è stata eseguita l'istruzione CNC, se lo stato Carry non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Carry è uguale a zero, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguità sarà quella con la label SUBR.

CNZ — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO. MA SOLTANTO SE LO STATO ZERO E' UGUALE A 0

CNZ
C4

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato Zero è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CNZ.

Si consideri la sequenza di istruzioni:



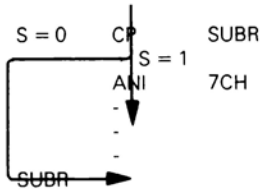
Dopo che è stata eseguita l'istruzione CNZ, se lo stato Zero non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Zero è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione che sarà eseguita è quella avente label SUBR.

CP — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO SIGN E' UGUALE A 0

CP
F4

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà chiamata solo se lo stato Sign è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CP.

Si consideri la sequenza di istruzioni:



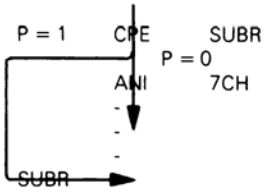
Dopo che è stata eseguita l'istruzione CP, sarà eseguita la istruzione ANI se lo stato Sign non è uguale a 0. Se lo stato Sign è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CPE — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO MA SOLTANTO SE LO STATO PARITA' E' UGUALE AD 1

CPE
EC

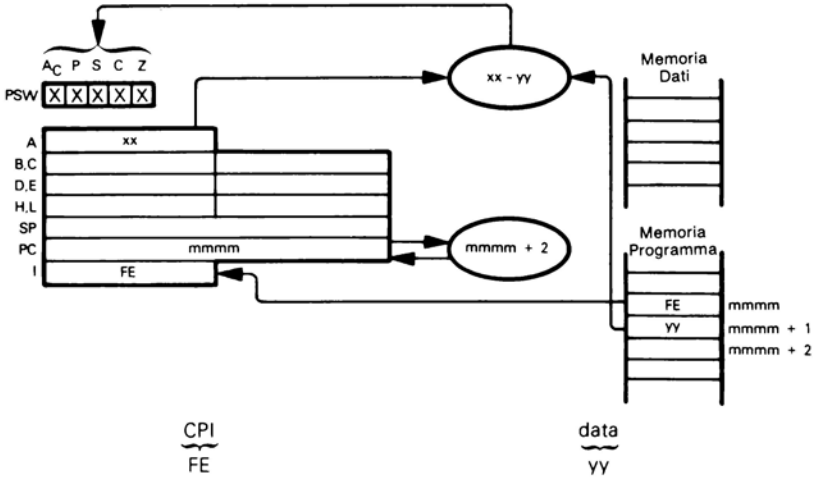
Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata può essere chiamata solo se lo stato Parità è uguale ad 1; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CPE.

Si consideri la sequenza di istruzioni:



Dopo che l'istruzione CPE è stata eseguita, se lo stato Parità non è uguale ad 1 sarà eseguita l'istruzione ANI. Se lo stato Parità è uguale ad 1, l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione ad essere eseguita sarà quella con la label SUBR.

CPI — CONFRONTA I CONTENUTI DELL'ACCUMULATORE CON DATI IMMEDIATI



Sottrae i contenuti del secondo byte in codice oggetto dai contenuti dell'Accumulatore trattando entrambi i numeri come singoli dati binari. Scarica il risultato, cioè lascia invariato l'Accumulatore, ma modifica i flag di Stato per riflettere il risultato della sottrazione.

Si supponga xx = $E3_{16}$ ed il secondo byte dell'istruzione CPI in codice oggetto contiene $A0_{16}$. Dopo che l'istruzione

CPI A0H

e stata eseguita l'Accumulatore conterrà $E3_{16}$ ma gli stati saranno modificati come segue:

```
                E3 =  11100011
Doppio complemento di A0 =  01100000
                            --------
                            01000011
```

1 — Tre bit 1, pone P a 0

Carry pone C a 0 — Risulta non-zero, pone Z a 0

0 pone S a 0 — No carry, così Ac è ripristinato a 0

Si noti che il Carry risultante è complementato.

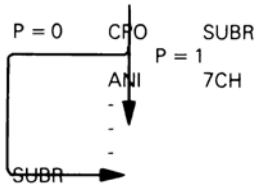
Questa è l'istruzione in gran parte usata per imporre gli stati precedenti l'esecuzione di un'istruzione di chiamata convenzionale, Ritorno e Salto.

CPO — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO, MA SOLTANTO SE LO STATO PARITA' E' UGUALE A 0

CPO
E4

Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà richiamata solo se lo stato Parità è uguale a 0; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CPO.

Si consideri la sequenza di istruzioni:



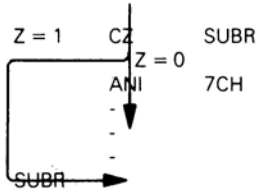
Dopo che è stata eseguita l'istruzione CPO, se lo stato Parità non è uguale a 0 sarà eseguita l'istruzione ANI. Se lo stato Parità è uguale a 0, l'indirizzo dell'istruzione ANI è conservato alla sommità dello stack. Il Puntatore dello Stack è decrementato di 2. La successiva istruzione ad essere eseguita sarà quella con la label SUBR.

CZ — CHIAMA LA SUBROUTINE IDENTIFICATA NELL'OPERANDO, MA SOLTANTO SE LO STATO ZERO E' UGUALE AD 1

CZ
CC

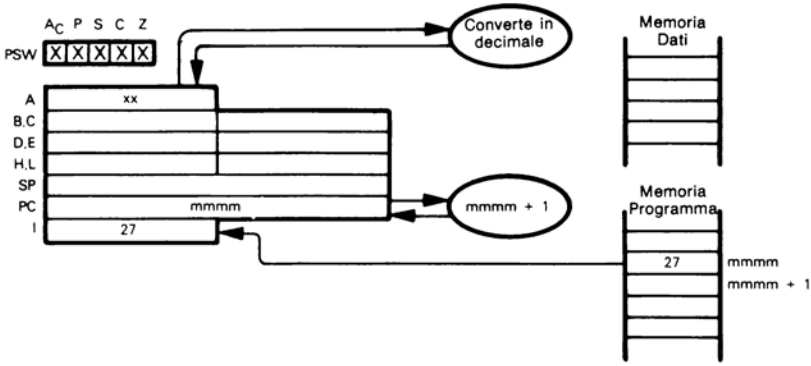
Questa istruzione è identica all'istruzione CALL eccetto che la subroutine identificata sarà richiamata solo se lo stato Zero è uguale a 1; diversamente sarà eseguita l'istruzione sequenzialmente seguente la CZ.

Si consideri la sequenza di istruzioni:



Dopo che l'istruzione CZ è stata eseguita, se lo stato Zero non è uguale a 1 sarà eseguita l'istruzione ANI. Se lo stato Zero è uguale ad 1 l'indirizzo dell'istruzione ANI è conservato alla sommità dello Stack. Il Puntatore dello Stack è decrementato di 2. La prossima istruzione eseguita sarà quella con la label SUBR.

DAA — AGGIUSTA I DECIMALI DELL'ACCUMULATORE



$$\underbrace{\text{DAA}}_{27}$$

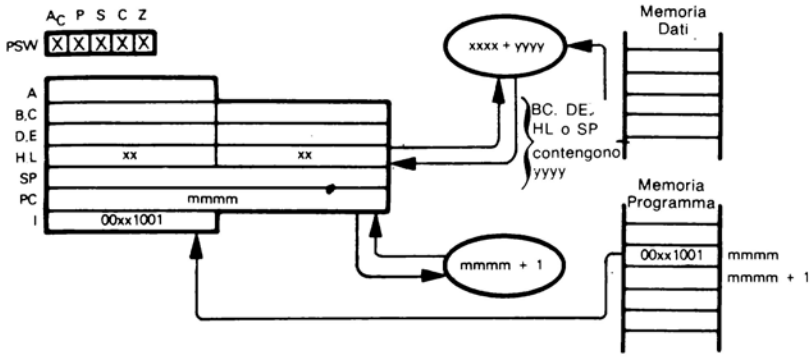
Converte i contenuti dell'Accumulatore nella loro forma decimale codificato binario. Questa istruzione potrebbe essere usata solo dopo l'addizione di due numeri BCD, cioè come si è visto ADD DAA oppure ADC DAA o SUB DAA o SBB DAA come istruzioni composte, aritmetiche decimali che operano in sorgente BCD per generare risposte BCD.

Si supponga che l'Accumulatore contenga 39_{16} ed il registro B contenga 47_{16}. Dopo che le istruzioni:

```
ADD  B
DAA
```

sono state eseguite l'Accumulatore conterrà 86_{16} e non 80_{16}. L'istruzione DAA modifica tutti i flag di stato, ma solo Carry è significativo.

DAD — SOMMA UNA COPPIA DI REGISTRI AD H ED L



0 0 per rp = B, rappresentante B, C
0 1 per rp = D, rappresentante D, E
1 0 per rp = H, rappresentante H, L
1 1 per rp = SP, rappresentante il Puntatore dello Stack

Somma il valore a 16 bit delle coppie BC, DE o HL oppure il Puntatore dello Stack alla coppia di registri HL.

Si supponga che H, L contenga $034A_{16}$ e B, C contenga $214C_{16}$. Dopo che l'istruzione

DAD B

è stata eseguita la coppia di registri HL conterrà 2496_{16}:

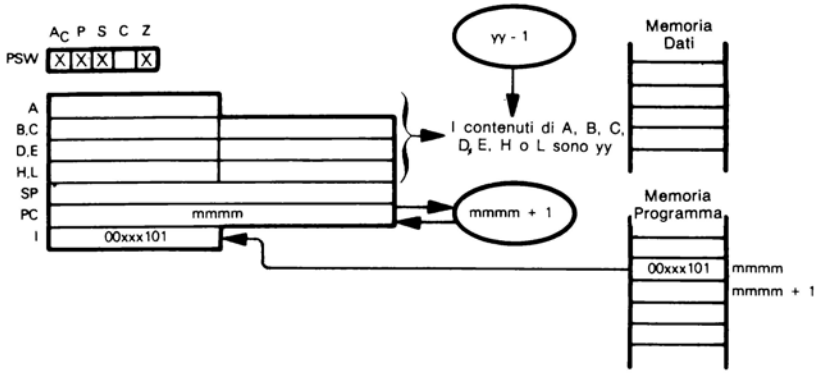
```
034A  =  0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0
214C  =  0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0
         ---------------------------------
         0 0 1 0 0 1 0 0 1 0 0 1 0 1 1 0
```

Non c'è carry così C è ripristinato a 0

Nessun altro stato è influenzato

L'istruzione DAD è una delle più usate nel set di istruzione dell'8080, per applicazioni di programmazione tradizionale. Questa istruzione fornisce l'equivalente di uno spostamento a sinistra di 16 bit.

DCR — DECREMENTA I CONTENUTI DI REGISTRO DI MEMORIA



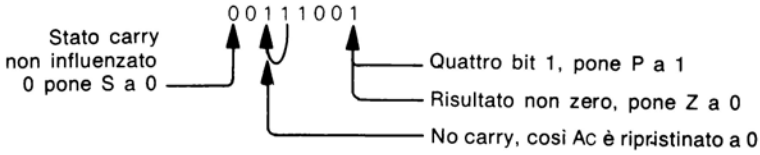
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae 1 dai contenuti del registro specificato.

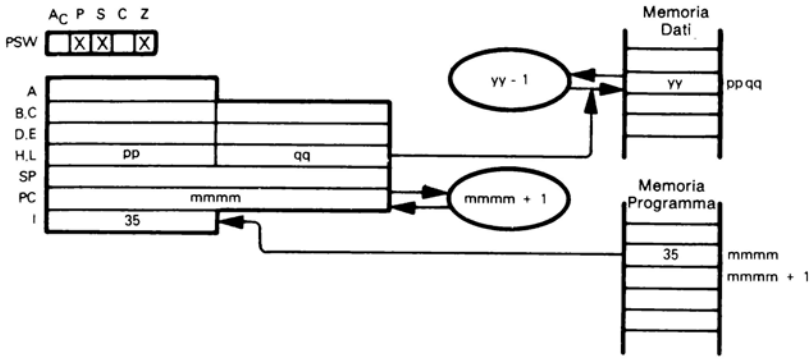
Si supponga che il registro C contenga $3A_{16}$. Dopo che l'istruzione

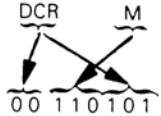
DCR C

è stata eseguita, il registro C conterrà 39_{16}:



Anche i contenuti del byte della memoria di lettura/scrittura possono essere decrementati:





Si supponga che HL contenga 3714_{16}. L'esecuzione dell'istruzione

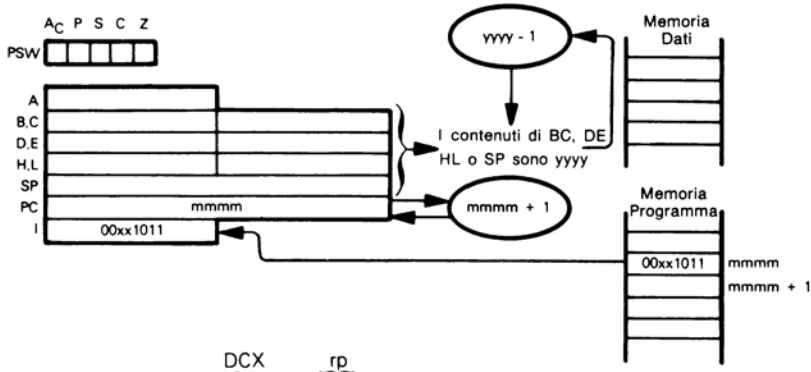
DCR M

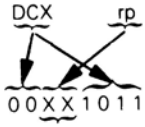
sottrae 1 dal contenuto del byte di memoria con indirizzo 3714_{16}. I flag di stato sono modificati come descritto per l'istruzione DCR C.

L'istruzione è usata in cicli di istruzione iterativi che impiegano un contatore dal valore minore uguale a 256. Una forma tipica di ciclo è la seguente:

```
        MVI     REG. DATI ; Carica il valore iniziale del contatore
LOOP    -                 ; Prima istruzione del ciclo
        —
        —
        —
        DCR     REG       ; Decrementa il contatore
        JNZ     LOOP      ; Ritorna se non zero
```

DCX — DECREMENTA UNA COPPIA DI REGISTRI





0 0 per rp = B, rappresentante B, C
0 1 per rp = D, rappresentante D, E
1 0 per rp = H, rappresentante H, L
1 1 per rp = SP, rappresentante il Puntatore dello Stack

Sottrae 1 dal valore a 16 bit contenuto nella coppia di registri specificata.

Si supponga che il Contatore dello Stack contenga $2F7A_{16}$. Dopo che l'istruzione

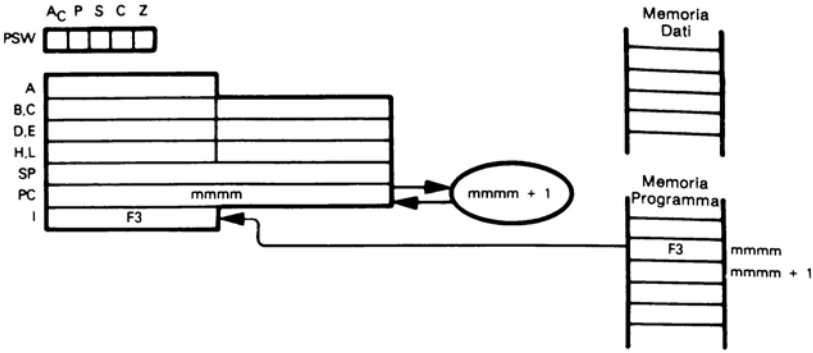
DCX SP

è stata eseguita, il contatore dello Stack conterrà $2F79_{16}$.

L'istruzione DCX non modifica nessun flag di stato e questo è un difetto nel set di istruzione dell'8080. Mentre l'istruzione DCR è usata nei cicli iterativi di istruzioni che usano un contatore con un valore minore o uguale a 256, l'istruzione DCX deve essere usata se il valore del contatore è maggiore di 256. Poichè l'istruzione DCX non influenza i flag di stato, deve essere semplicemente aggiunta al testo un'altra istruzione per il risultato zero. Una forma tipica di ciclo è la seguente:

```
        LXI     D, DATI     ; Carica il valore iniziale a 16 bit nel contatore
LOOP    —       —           ; Prima istruzione del ciclo
        —       —
        —       —
        —       —
        DCX     D           ; Decrementa il contatore
        MOV     A,D         ; Per il test per zero, muove D ad A
        ORA     E           ; Quindi OR di A con E
        JNZ     LOOP        ; Ritorna se non zero
```

DI – DISABILITA INTERRUPT

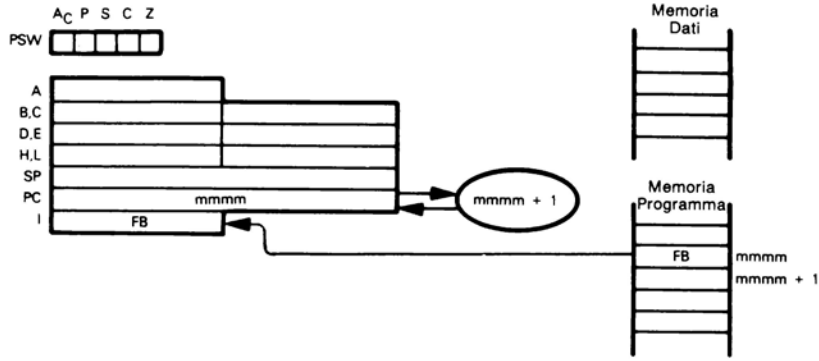


$$\underbrace{\text{DI}}_{\text{F3}}$$

Dopo che è stata eseguita questa istruzione il segnale INTE è basso all'uscita della CPU dell'8080 e non verrà accettata nessuna richiesta di interrupt. Nessun registro o flag è influenzato.

Si ricordi che quando un interrupt è accettato, gli interrupt sono automaticamente disabilitati.

EI – ABILITA INTERRUPT



$$\underbrace{\text{EI}}_{\text{FB}}$$

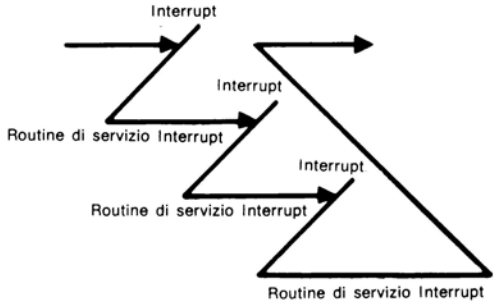
Quando è eseguita questa istruzione, gli interrupt sono abilitati a partire dall'istante in cui è terminata l'istruzione che si stava eseguendo.

La maggior parte delle routines di servizio interrupt terminano con le due istruzioni:

```
EI          ; Abilita gli interrupt
RET         ; Ritorna al programma interrotto
```

Se gli interrupt sono eseguiti sequenzialmente, allora per l'intera durata della routine di servizio degli interrupt, tutti gli interrupt sono disabilitati — ciò significa che in una applicazione a multi-interrupt c'è una significativa possibilità per uno o più interrupt di essere in sospeso fino a quando una qualsiasi routine di servizio interrupt completa l'esecuzione.

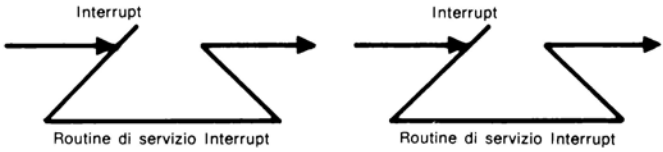
Se gli interrupt erano ammessi non appena l'istruzione EI è stata eseguita, allora l'istruzione di ritorno non sarà eseguita. In queste circostanze gli interrupt potrebbero impilarsi uno sull'altro e consumare dello Stack di memoria non necessario. Questo può essere illustrato come segue:

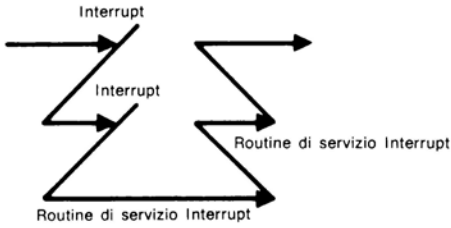


Inibendo gli interrupt per più istruzioni seguenti l'esecuzione di EI, la CPU dell'8080 assicura che l'istruzione RET è eseguita con la seguenza:

```
—
—
—
EI              ; Abilita gli interrupt
RET             ; Ritorno da interrupt
```

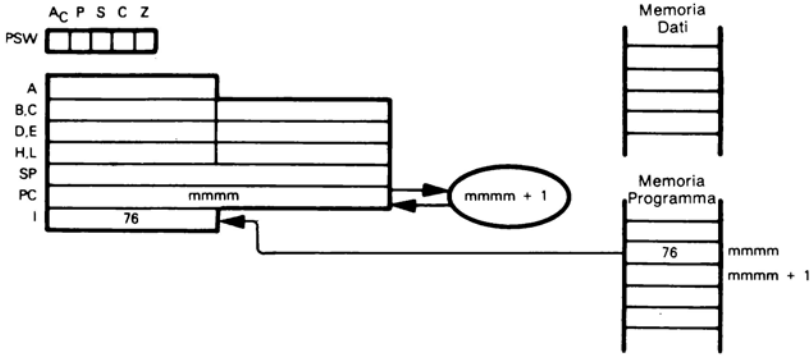
E' abbastanza comune per gli interrupt non essere abilitati mentre si sta eseguendo una routine di servizio interrupt. Gli interrupt sono processari in serie:





Se gli interrupt sono processati in serie la regolazione di priorità sarà applicata soltanto durante i processi di ammissione.

HLT – ARRESTO

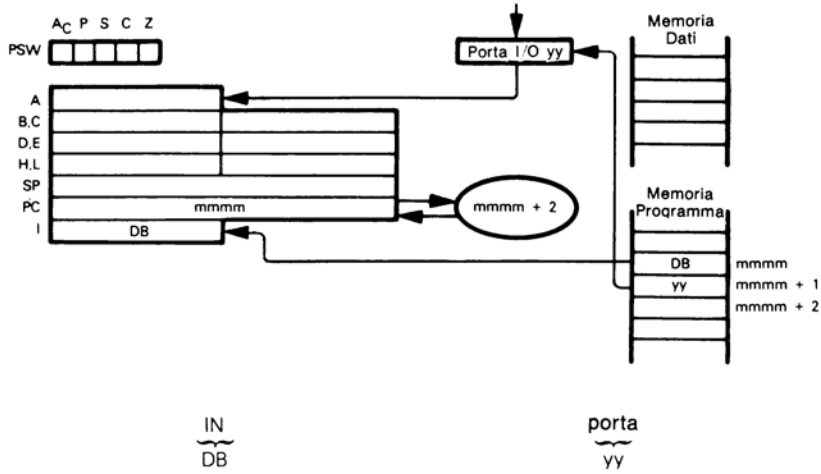


HLT
76

Quando viene eseguita l'istruzione HLT cessa l'esecuzione del programma. Per fare ripartire l'esecuzione si richiede un interrupt od un reset. Nessun registro o stato viene influenzato.

NOTA: Gli interrupt non sono abilitati da una istruzione EI precedente l'istruzione HALT, la CPU dell'8080 non può uscire dallo stato HALT se non mediante attivazione tramite ripristino hardware.

IN – INGRESSO ALL'ACCUMULATORE



Carica un byte nell'Accumulatore della porta I/O identificata dal secondo byte della istruzione in codice oggetto IN.

Si supponga che il Buffer della Porta I/O $1A_{16}$ contenga 36_{16}. Dopo che l'istruzione:

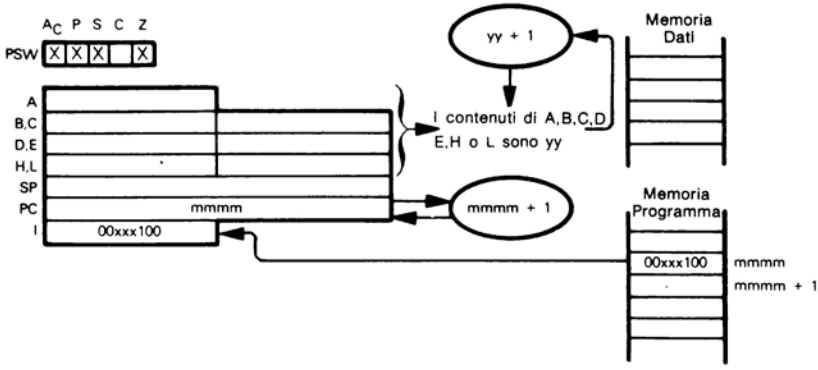
IN 1AH

è stata eseguita l'Accumulatore conterrà 36_{16}.

L'istruzione IN non influenza nessuno stato.

L'uso dell'istruzione IN dipende molto dall'hardware. La validità di indirizzi per porte I/O è determinata dal modo in cui è stata realizzata la logica I/O. E' anche possibile progettare un sistema a microcomputer impiegante istruzione di reference della memoria con specifici indirizzi di memoria.

INR – INCREMENTA REGISTRO O CONTENUTI DI MEMORIA



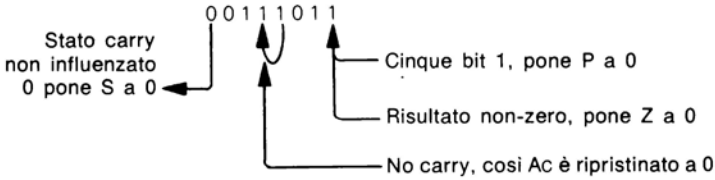
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Aggiunge 1 ai contenuti del registro specificato.

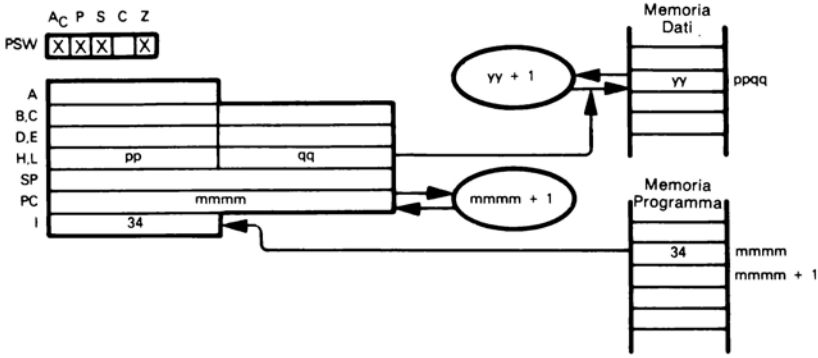
Si supponga che il registro C contenga $3A_{16}$. Dopo che l'istruzione:

INR C

è stata eseguita il registro conterrà $3B_{16}$:



Anche i contenuti del byte della memoria di lettura/scrittura possono essere incrementati.



Si supponga che HL contenga 3714_{16}. Allora l'esecuzione dell'istruzione

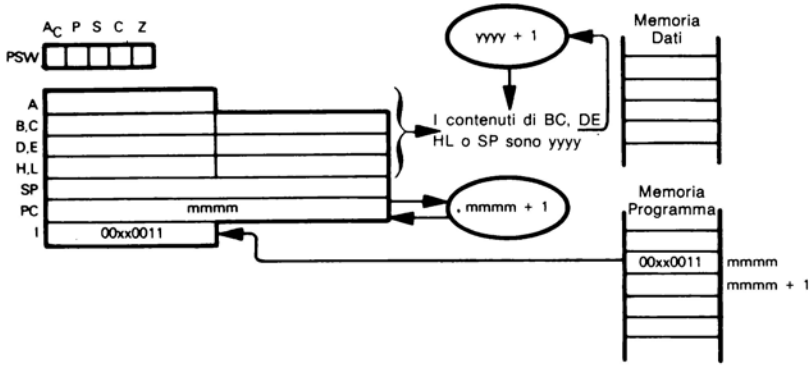
INR M

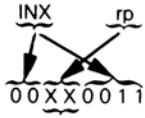
aggiunge 1 ai contenuti del byte di memoria con l'indirizzo 3714_{16}. I flag di stato sono modificati come descritto per la istruzione INR C.

L'istruzione INR è usata nei cicli iterativi di istruzioni che usano un contatore di valore minore uguale a 256. Una forma tipica di ciclo è la seguente:

```
        MVI     REG, DATI ; Carica il commento del valore iniziale del contatore
LOOP    —       —         ; Prima istruzione del ciclo
        —       —
        —       —
        INR     REG       ; Decrementa il contatore
        JNZ     LOOP      ; Ritorna se non zero
```

INX – INCREMENTA COPPIA DI REGISTRI





0 0 per rp = B, rappresentante B, C
0 1 per rp = D, rappresentante D, E
1 0 per rp = H, rappresentante H, L
1 1 per rp = SP, rappresentante il Puntatore dello Stack

Somma 1 al valore a 16 bit contenuto nella coppia di registri specificata.

Supponendo che i registri D ed E contengano $2F7A_{16}$. Dopo che l'istruzione

INX D

è stata eseguita, i registri D ed E conterranno $2F7B_{16}$.

L'istruzione INX non modifica nessun flag di stato, questo è un difetto nel set di istruzioni dell'8080. Mentre l'istruzione DCR è usata nei cicli iterativi di istruzione che usano un contatore con valore minore uguale di 256, l'istruzione INX deve essere usata se il valore del contatore è maggiore di 256. Poichè l'istruzione INX non impone flag di stato, occorre semplicemente aggiungere altre istruzioni per il test di risultato zero. Una forma tipica di ciclo è la seguente:

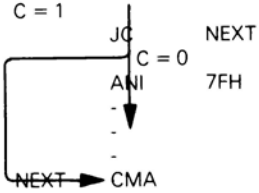
```
        LXI     D,DATI    ; Carica il complemento a 16 bit del valore iniziale
                          ; del contatore
LOOP    —       —         ; Prima istruzione del ciclo
        —       —
        —       —
        INX     D         ; Incrementa il contatore
        MOV     A,D       ; Per il test per zero, muovi D ad A
        ORA     E         ; Quindi OR di A con E
        JNZ     LOOP      ; Ritorna se non zero
```

JC — SALTA SE CARRY

JC
DA

Questa istruzione è identica all'istruzione JMP solo che qui l'istruzione di Salto è eseguita solo se lo stato Carry è uguale ad 1; altrimenti viene eseguita l'istruzione sequenzialmente successiva.

Si osservi la seguente sequenza di istruzioni:



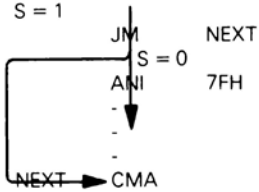
Dopo l'istruzione JC viene eseguita la CMA, se lo stato Carry è uguale ad 1. L'istruzione ANI viene eseguita se lo stato Carry è uguale a 0.

JM — SALTA SE MENO

JM
FA

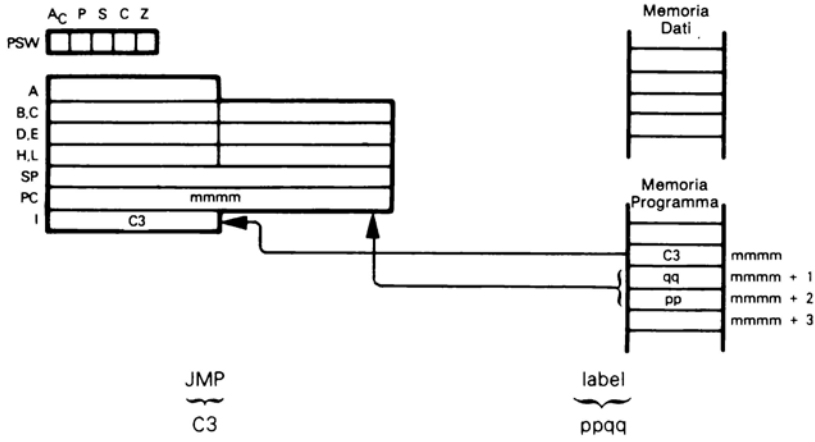
Questa istruzione è identica alla JMP eccetto che il salto è eseguito solo se lo stato Sign è uguale ad 1; altrimenti verrà eseguita la successiva istruzione.

Si osservi la seguente sequenza di istruzioni:



Dopo l'istruzione JM viene eseguita l'istruzione CMA se lo stato Sign è uguale ad 1. L'istruzione ANI è eseguita se lo stesso Sign è uguale a 0.

JMP – SALTA ALL'ISTRUZIONE IDENTIFICATA NELL'OPERANDO



Carica i contenuti del secondo e terzo byte dell'istruzione di Salto in codice oggetto nel Contatore di Programma; questo diventa l'indirizzo di memoria per la successiva istruzione da eseguire. Il precedente contenuto del Contatore di Programma viene perso.

Si osservi la seguente sequenza di istruzioni:

```
        JMP     NEXT
        ANI     7FH
        –
        –
        –
NEXT    CMA
```

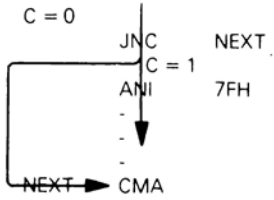
Dopo l'istruzione JMP sarà eseguita l'istruzione CMA. L'istruzione ANI non verrà eseguita fino a che una qualche istruzione di salto della sequenza di istruzione non opera un salto a questa istruzione.

JNC – SALTA SE NON CARRY

JNC
D2

Questa istruzione è identica all'istruzione JMP eccetto che il salto è eseguito solo se lo stato Carry è uguale a 0; diversamente viene eseguita l'istruzione successiva.

Si osservi la seguente sequenza di istruzioni:



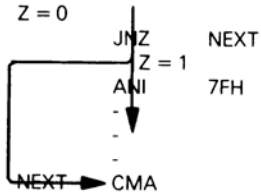
Dopo l'istruzione JNC viene eseguita l'istruzione CMA se lo stato Carry è uguale a 0. L'istruzione ANI viene eseguita se lo stato Carry è uguale ad 1.

JNZ — SALTA SE NON ZERO

JNZ
C2

Questa istruzione è identica alla JMP eccetto che il salto viene eseguito solo se lo stato Zero è uguale a 0; altrimenti viene eseguita l'istruzione successiva.

Si osservi la seguente sequenza di istruzioni:



Dopo l'istruzione JNZ viene eseguita la CMA se lo stato Zero è uguale a 0.

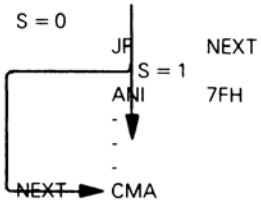
L'istruzione ANI viene eseguita se lo stato Zero è uguale ad 1.

JP — SALTA SE POSITIVO

JP
F2

Questa istruzione è identica alla JMP tranne il fatto che il salto viene eseguito solo se lo stato Sign vale 0; altrimenti viene eseguita l'istruzione successiva.

Si osservi la seguente sequenza di istruzioni:



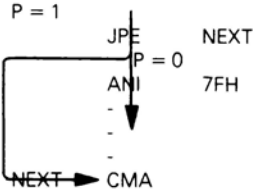
Dopo l'istruzione JP viene eseguita la CMA se lo stato Sign è uguale a 0. L'istruzione ANI viene eseguita se lo Stato Sign è uguale ad 1.

JPE — SALTA SE PARITA' PARI

JPE
EA

Questa istruzione è identica alla JMP tranne il fatto che il salto viene eseguito se lo stato Parità è uguale ad 1; altrimenti viene eseguita l'istruzione successiva.

Si osserva la seguente sequenza di istruzioni:



Dopo l'istruzione JPE viene eseguita la CMA se lo stato Parità è uguale ad 1.

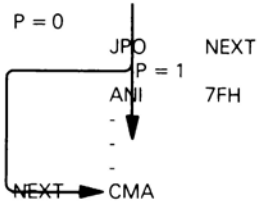
L'istruzione ANI viene eseguita se lo stato Parità è uguale a 0.

JPO — SALTA SE PARITA' DISPARI

JPO
E2

Questa istruzione è identica alla JMP tranne il fatto che il salto è eseguito solo se lo stato Parita vale 0; altrimenti viene eseguita l'istruzione successiva.

Si osservi la seguente sequenza di istruzioni:



Dopo l'istruzione JPO viene eseguita la CMA se lo stato Parità è uguale a 0.

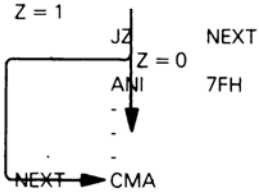
L'istruzione ANI viene eseguita se lo stato Parità è uguale ad 1.

JZ — SALTA SE ZERO

JZ
CA

Questa istruzione è identica alla JMP tranne il fatto che il salto è eseguito solo se lo stato Zero è uguale ad 1; altrimenti viene eseguita la prossima istruzione.

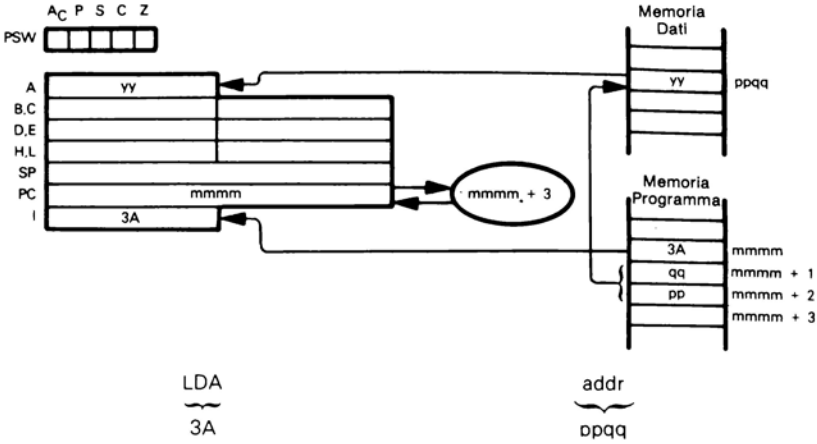
Si osservi la seguente sequenza di istruzioni:



Dopo l'istruzione JZ viene eseguita la CMA se lo stato Zero è uguale ad 1.

L'istruzione ANI viene eseguita se lo stato Zero è uguale a 0.

LDA — CARICA L'ACCUMULATORE DELLA MEMORIA UTILIZZANDO L'INDIRIZZAMENTO DIRETTO



Carica nell'Accumulatore il contenuto del byte di memoria indirizzato direttamente dal secondo e terzo byte dell'istruzione LDA in codice oggetto.

Si supponga che il byte di memoria $084A_{16}$ contenga $3A_{16}$. Dopo che l'istruzione:

```
LABEL   EQU     084AH
        -
        -
        -
        LDA     LABEL
```

è stata eseguita l'Accumulatore conterrà $3A_{16}$.

Si ricordi che EQU è un Ordine Assembler e non una istruzione; esso impone l'uso del valore a 16 bit $084A_{16}$ dovunque appare LABEL. L'istruzione:

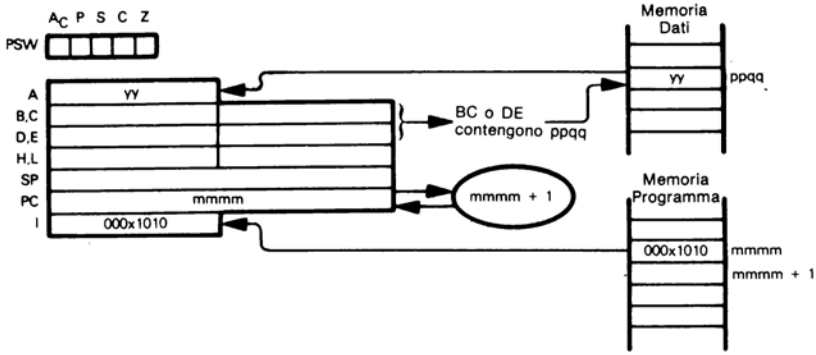
```
LDA   LABEL
```

è equivalente alle due istruzioni:

```
LXI     H,LABEL
MOV     A,M
```

L'istruzione LDA è preferita quando si sta caricando un singolo dato dalla memoria; essa usa una istruzione e tre byte in programma oggetto per fare quello che fa la combinazione LXI MOV in due istruzioni e quattro byte in programma oggetto. Inoltre la combinazione LXI MOV usa i registri H ed L; l'istruzione LDA invece no.

LDAX – CARICA L'ACCUMULATORE DALLA LOCAZIONE DI MEMORIA INDIRIZZATA MEDIANTE UNA COPPIA DI REGISTRI



0 se rp è B, rappresentante BC
1 se rp è D, rappresentante DE

Carica nell'Accumulatore i contenuti del byte di memoria indirizzato dalla coppia di registro BC o DE.

Si supponga che il registro B contenga 08_{16}, il registro C contenga $4A_{16}$ ed il byte di memoria $084A_{16}$ contenga $3A_{16}$. Dopo che l'istruzione:

LDAX B

è stata eseguita, l'Accumulatore conterrà $3A_{16}$.

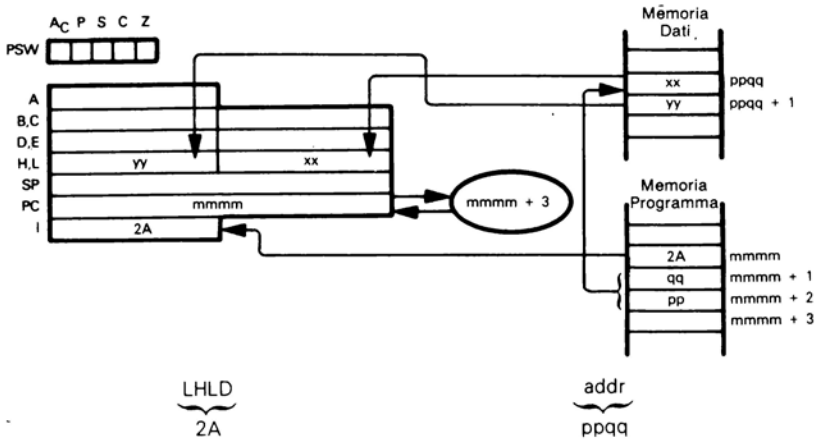
Si noti che non esiste l'istruzione LDAX H poichè è identica all'istruzione MOV A,M.

Le istruzioni LDAX ed LXI saranno usate normalmente assieme, poichè l'istruzione LXI carica un indirizzo a 16 bit nei registri BC o DE come segue:

```
LXI     B,084AH
LDAX    B
```

Si noti che l'istruzione LDAX caricherà dati soltanto nell'Accumulatore mentre la istruzione MOV caricherà dati in qualsiasi registro.

LHLD — CARICA DIRETTAMENTE I REGISTRI H ED L



Il secondo e terzo byte in codice oggetto forniscono l'indirizzo di memoria di un byte di dati, il cui contenuto sarà caricato nel registro L. Il contenuto del byte dati sequenzialmente successivo è caricato nel registro H.

Si supponga che il byte di memoria $084A_{16}$ contenga $3A_{16}$ ed il byte $084B_{16}$ contenga $2C_{16}$. Dopo che l'istruzione:

```
LABEL   EQU     084AH
        -
        -
        -
        LHLD    LABEL
```

è stata eseguita, il registro H conterrà $2C_{16}$ ed il registro L conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assembler e non una istruzione: esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ dove si trova LABEL.

L'istruzione LHLD è una versione ad indirizzamento diretto della LXI H,DATI. Per esempio l'istruzione:

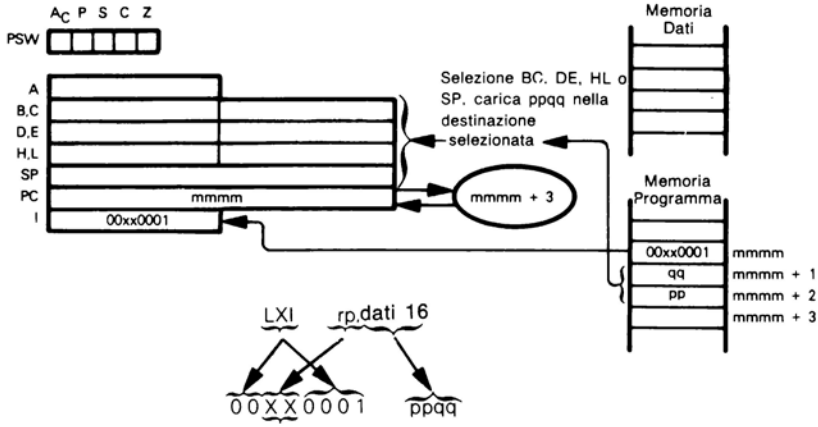
```
LXI   H,2C3AH
```

caricherà $2C_{16}$ nel registro H e $3A_{16}$ nel registro L.

Per i dati a 16 bit non soggetti a variazioni, si usi LXI A, DATI al posto di LHLD ADDR.

Si ricordi, se IND indirizza direttamente un byte della memoria di lettura/scrittura, si può cambiare il valore che sarà caricato nei registri H ed L mediante una istruzione LHLD. Per fare questo si scriva semplicemente il nuovo valore in ADDR ed ADDR+1.

LXI – CARICA UN VALORE A 16 BIT, IMMEDIATO IN UNA COPPIA DI REGISTRI



0 0 se rp = B, selezionando i registri B e C
0 1 se rp = D, selezionando i registri D ed E
1 0 se rp = H, selezionando i registri H ed L
1 1 se rp = SP selezionando il Puntatore dello Stack

Carica nella coppia di registri selezionati i contenuti del secondo e terzo byte in codice oggetto.

Dopo che l'istruzione:

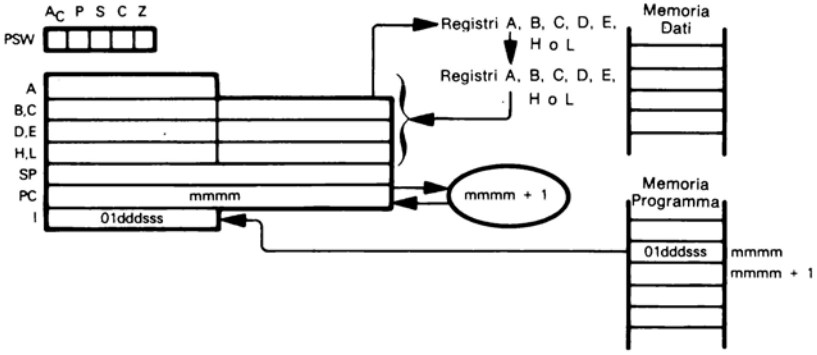
LXI SP,217AH

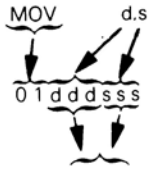
è stata eseguita il Puntatore dello Stack conterrà $217A_{16}$.

LXI è l'istruzione più spesso usata per il caricamento di indirizzi in una coppia di registri.

MOV – MUOVI DATI

Questa istruzione assume due forme. La prima opera il trasferimento dei contenuti di un registro in un altro.





0 0 0 d o s è il Registro B
0 0 1 d o s è il Registro C
0 1 0 d o s è il Registro D
0 1 1 d o s è il Registro E
1 0 0 d o s è il Registro H
1 0 1 d o s è il Registro L
1 1 1 d o s è l'Accumulatore

Muove i contenuti di qualsiasi registro in qualunque altro. Per esempio:

MOV A,B

muove i contenuti del registro B nell'Accumulatore

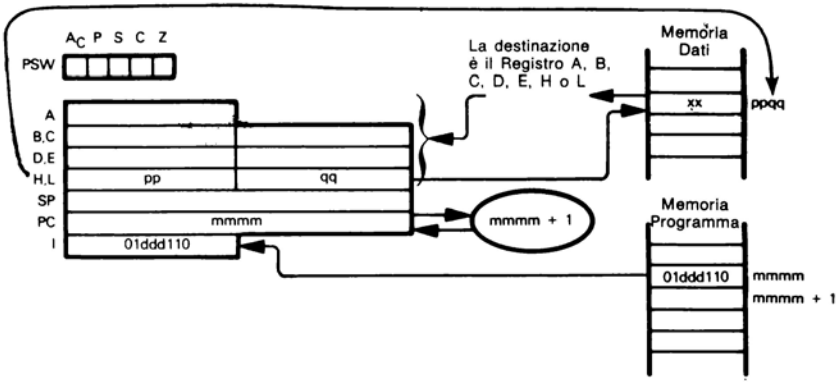
MOV L,D

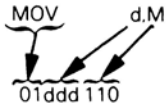
muove i contenuti del registro D nel registro L.

MOV C,C

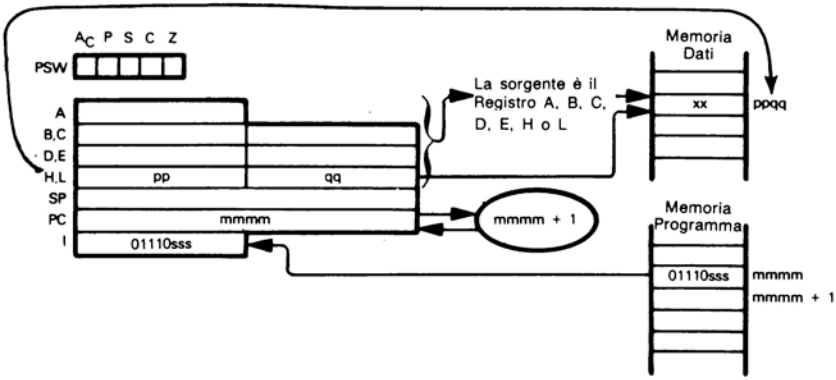
non succede nulla in quanto il registro C compare come sorgente e destinazione.

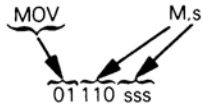
La sorgente dei dati può anche esere un byte di memoria:





Oppure un byte di memoria può essere la destinazione dei dati:





In entrambi i casi ddd o sss è il registro sul quale viene operato il movimento dati.

Così l'istruzione

MOV M,A

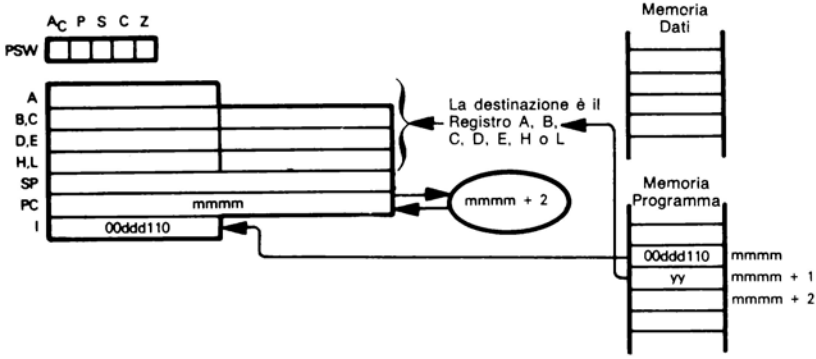
muove il contenuto dell'Accumulatore nel byte di memoria di lettura/scrittura indirizzato mediante i registri H ed L. L'istruzione:

MOV L,M

muove i contenuti del byte di memoria indirizzato mediante i registri H ed L nel registro L.

L'istruzione di Movimento nelle sue varie forme è l'istruzione usata più frequentemente per l'8080.

MVI — MUOVI DATI IN MODO IMMEDIATO IN UN REGISTRO O IN MEMORIA



000 per reg. = B
001 per reg. = C
010 per reg. = D
011 per reg. = E
100 per reg. = H
101 per reg. = L
111 per reg. = A

Muove i contenuti del secondo byte in codice oggetto ad uno dei registri.

Quando viene eseguita l'istruzione:

```
MVI   A,2AH
```

viene caricato $2A_{16}$ nell'Accumulatore. L'istruzione:

```
MVI   H,03H
```

carica 03_{16} nel registro H.

Le istruzioni di caricamento immediato dati su registro sono usate molto frequentemente nei programmi 8080.

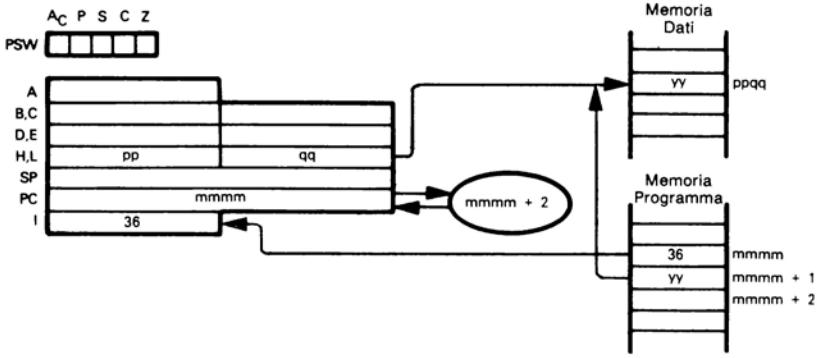
Si noti che l'istruzione LXI è equivalente a due istruzioni MVI; per esempio:

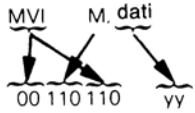
```
LXI   H,032AH
```

è equivalente a:

```
MVI     H,03H
MVI     L,2AH
```

I dati possono anche essere caricati immediatamente in un byte della memoria di lettura/scrittura:





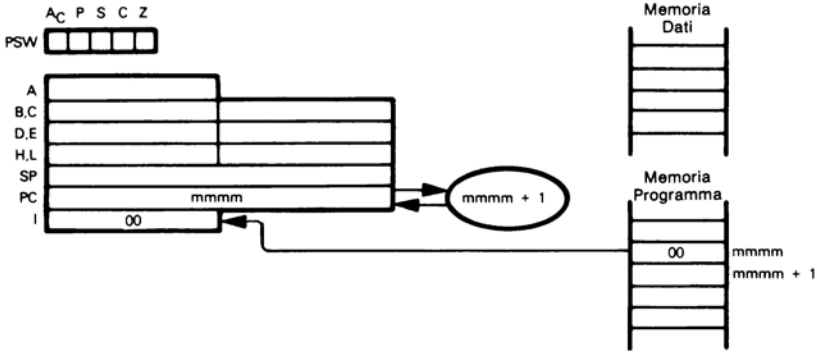
Si supponga che il registro H contenga 03_{16} ed il registro L contenga $2A_{16}$; quando viene eseguita l'istruzione:

MVI M,2CH

$2C_{16}$ sarà caricato nel byte di memoria $032A_{16}$.

L'istruzione di caricamento immediato in memoria (MVI M,dati) è usata molto meno dell'istruzione di caricamento immediato in Registro (MVI reg.dati).

NOP — NON OPERARE



NOP
00

Quando viene eseguita questa istruzione non accade nulla; essa è presente per tre ragioni:

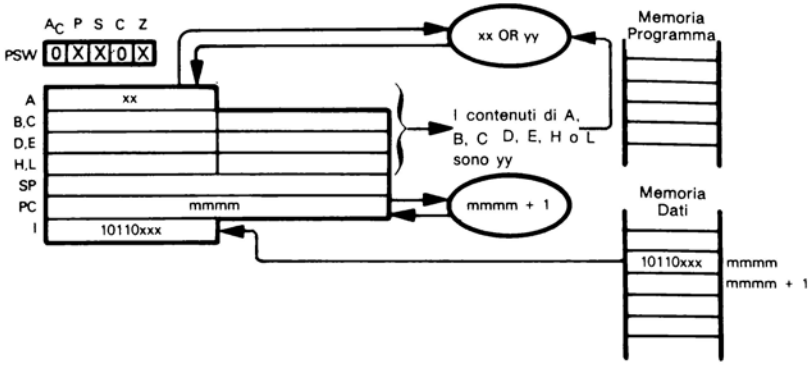
1) Quando è presente un errore di programma, che condurrebbe alla ricerca di un codice oggetto da una memoria non esistente, viene assegnato come codice oggetto 00. Questo è un buon artificio per assicurare l'eliminazione degli errori più comuni dei programmi.

2) L'istruzione NOP permette di ricavare una label da un byte in programma oggetto:

HERE NOP

3) Per un buon accordo dei tempi di ritardo. Ogni istruzione NOP aggiunge quattro cicli di clock ad un ritardo. L'istruzione NOP non è molto pratica né usata frequentemente.

ORA — OR DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE



000 per reg. = B
001 per reg. = C
010 per reg. = D
011 per reg. = E
100 per reg. = H
101 per reg. = L
111 per reg. = A

Opera l'OR dei contenuti dell'Accumulatore con qualunque altro registro. Il risultato è memorizzato nell'Accumulatore.

Si supponga xx = $E3_{16}$, il registro E contenga $A8_{16}$. Dopo che l'istruzione:

ORA E

è stata eseguita, l'Accumulatore conterrà EB_{16}.

```
E3 = 11100011
A8 = 10101000
     --------
     11101011
```

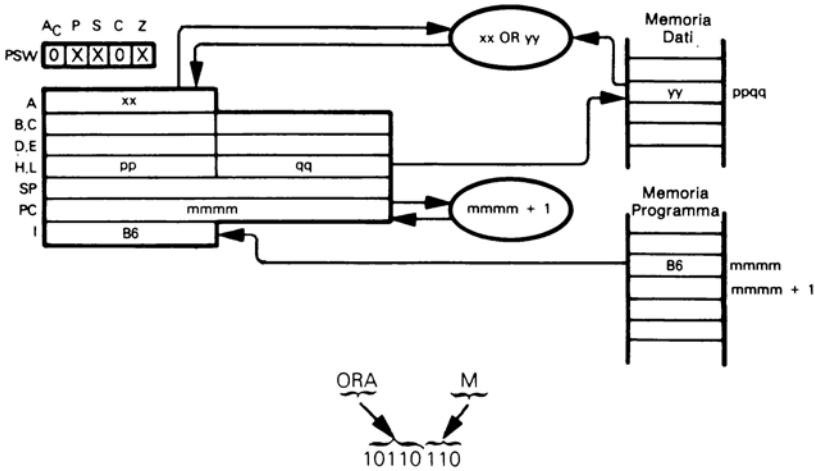
Carry è sempre posto a 0

1 pone S ad 1

Sei bit 1, pone P ad 1

Risultato non-zero, pone Z a 0

Anche i contenuti del byte di memoria possono essere usati per operare l'OR con l'Accumulatore:



Se xx = $E3_{16}$ ed yy = $A8_{16}$ allora l'esecuzione dell'istruzione:

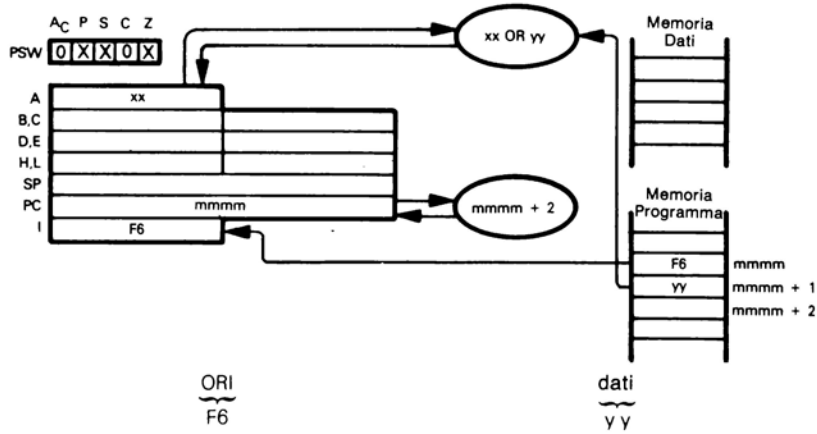
ORA M

genera lo stesso risultato dell'esecuzione dell'istruzione ORA E, appena descritta.

L'istruzione ORA non viene usata così frequentemente come l'istruzione di OR immediato (ORI).

Si noti che operando l'OR dell'Accumulatore con sé stesso (ORA A) permette di azzerare lo stato Carry; questa istruzione viene impiegata per porre gli stati dopo le istruzioni INX e DCX.

ORI – OR IMMEDIATO CON L'ACCUMULATORE



Opera l'OR dell'Accumulatore con i contenuti del secondo byte di istruzione in codice oggetto.

Si supponga xx = $3A_{16}$. Dopo che è stata eseguita l'istruzione:

ORI 7CH

l'Accumulatore conterrà $7E_{16}$:

```
3A = 00111010
7C = 01111100
     --------
     01111110
```

Carry è sempre posto a 0

0 pone S a 0

6 bit 1, pone P ad 1

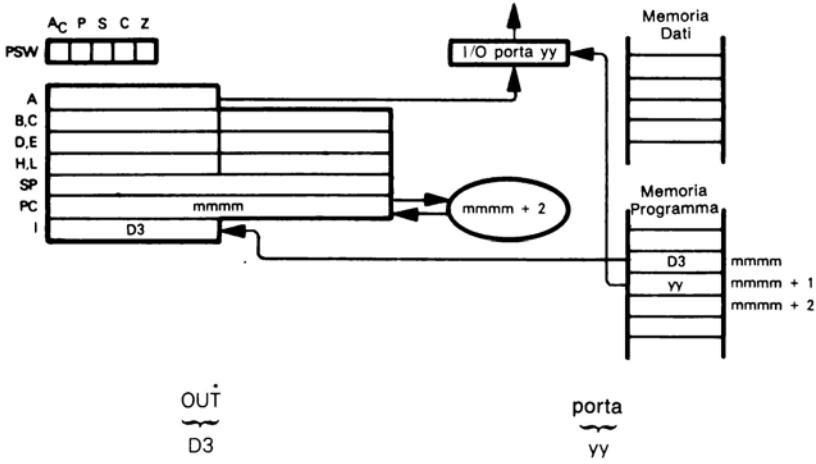
Risultato non-zero, pone Z a 0

Questa è una istruzione logica di routine; viene spesso usata per commutare "on" dei bit. Per esempio l'istruzione

ORI 80H

porrà incondizionatamente ad 1 il bit di ordine più elevato dell'Accumulatore.

OUT — USCITA DALL'ACCUMULATORE



Preleva i contenuti dall'Accumulatore e li fornisce alla porta I/O identificata dal secondo byte in codice oggetto dell'istruzione OUT.

Si supponga che l'Accumulatore contenga 36_{16}. Dopo che l'istruzione:

OUT 1AH

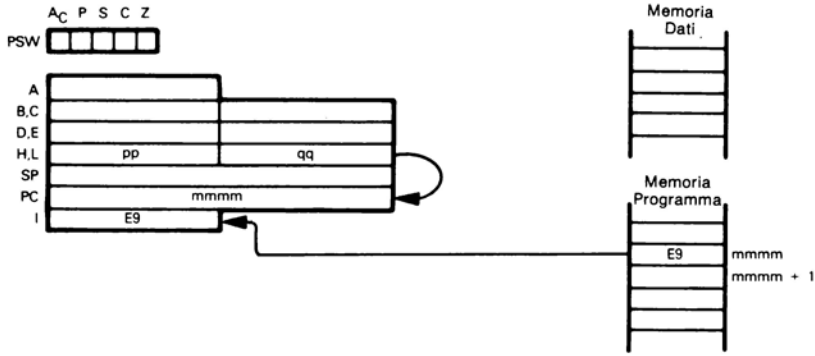
è stata eseguita, 36_{16} sarà stato trasferito nel buffer della porta I/O $1A_{16}$.

L'istruzione OUT non influenza nessuno stato.

L'uso dell'istruzione OUT dipende molto dall'hardware. Validi indirizzi della porta I/O sono determinati dal modo in cui è stata realizzata la logica I/O. E' anche possibile progettare un sistema a microcomputer che acceda alla logica esterna mediante istruzioni reference della memoria con specifici indirizzi di memoria.

L'istruzione OUT è spesso impiegata particolarmente per il controllo della logica esterna alla CPU del microcomputer.

PCHL — SALTA ALL'INDIRIZZO SPECIFICATO MEDIANTE HL



PCHL
E9

I contenuti dei registri H ed L vengono inviati al Contatore di Programma quindi viene eseguito un salto ad indirizzamento implicito.

La sequenza di istruzioni:

```
LXI     H,ADDR
PCHL
```

ha esattamente lo stesso effetto finale della singola istruzione:

```
JMP     ADDR
```

Entrambe specificano che la istruzione da eseguire è quella con la label ADDR.

L'istruzione PCHL è comoda quando si vuole incrementare un indirizzo di ritorno di una subroutine avente ritorni multipli.

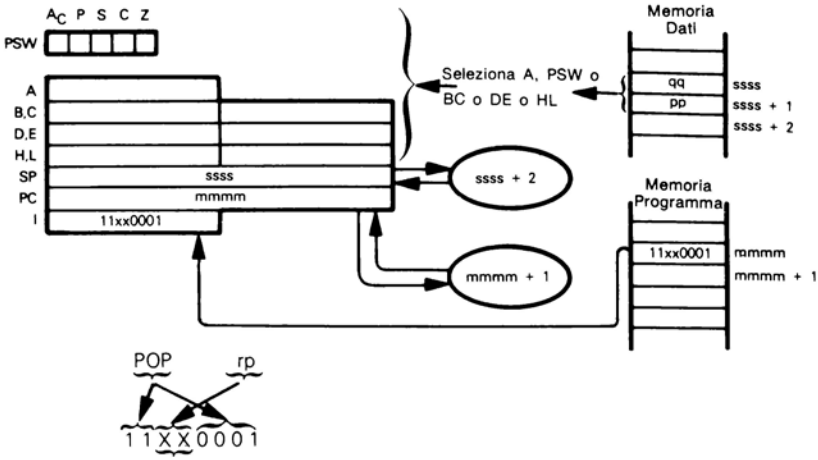
Si consideri la seguente chiamata della subroutine SUB:

```
CALL    SUB     ; Chiama subroutine
JMP     ERR     ; Ritorno errato
                ; Ritorno esatto
```

Usando RET per ritornare dalla SUB il ritorno potrebbe essere operato dall'esecuzione della JMP ERR; perciò se la SUB viene eseguita senza rivelare condizioni di errore il ritorno avviene come segue:

```
POP     H       ; Poni l'indirizzo di ritorno in HL
INX     H       ; Somma 3 all'indirizzo di ritorno
INX     H
INX     H
PCHL            ; Ritorno
```

POP – LEGGI DALLA SOMMITA' DELLO STACK



0 0 se rp è B, selezionando i registri B e C
0 1 se rp è D, selezionando i registri D ed E
1 0 se rp è H, selezionando i registri H ed L
1 1 se rp è PSW, selezionando l'Accumulatore ed i flag di stato come una unità a 16 bit

Pone i due byte della sommità dello stack nella coppia di registri specificata.

Si supponga qq = 03_{16} e pp = $2A_{16}$. L'esecuzione dell'istruzione:

POP H

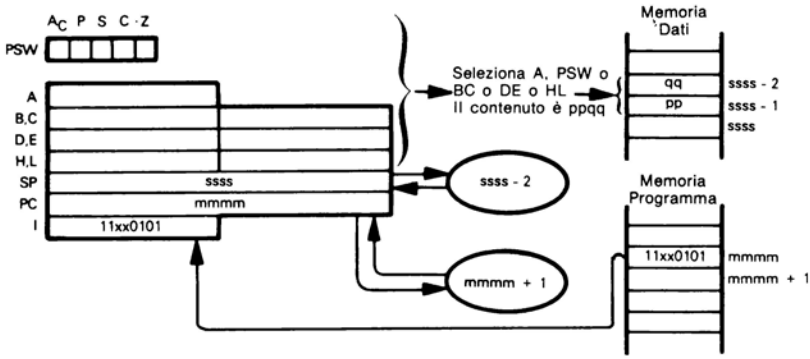
carica 03_{16} nel registro L e $2A_{16}$ nel registro H. L'esecuzione dell'istruzione:

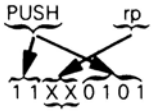
POP PSW

carica 03_{16} nei flag di stato e $2A_{16}$ nell'Accumulatore. Così lo stato C sarà posto ad 1 e gli altri stati saranno azzerati.

L'istruzione POP è usata essenzialmente per rimemorizzare i contenuti di registri e stati che erano stati conservati nello stack, per esempio, mentre si stava eseguendo un interrupt.

PUSH — SCRIVI NELLA SOMMITA' DELLO STACK





0 0 se rp è B, selezionando i registri B e C
0 1 se rp è D, selezionando i registri D ed E
1 0 se rp è H, selezionando i registri H ed L
1 1 se rp è PSW, selezionando l'Accumulatore ed i flag di stato come unità a 16 bit

Manda i contenuti della coppia di registri specificata alla sommità dello stack.

Si supponga che il registro H contenga 03_{16} ed L contenga $2A_{16}$. L'esecuzione della istruzione:

PUSH H

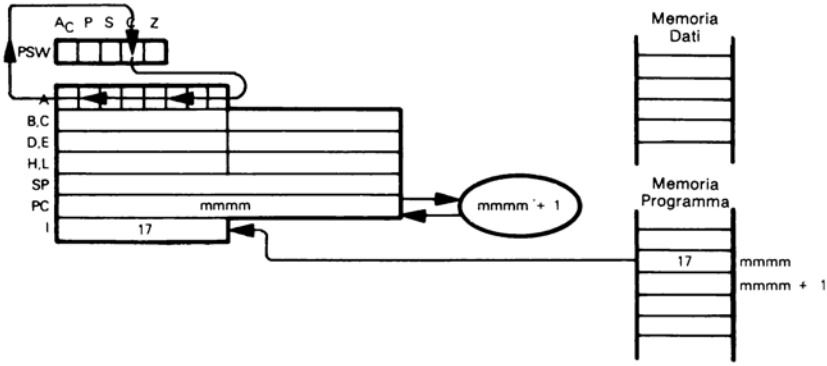
carica 03_{16} e poi $2A_{16}$ alla sommità dello Stack. L'esecuzione dell'istruzione:

PUSH PSW

carica l'Accumulatore e poi i flag di stato alla sommità dello Stack.

L'istruzione PUSH è usata essenzialmente per conservare i contenuti di registri e stati, per esempio, prima dell'esecuzione di un interrupt.

RAL — RUOTA L'ACCUMULATORE A SINISTRA ATTRAVERSO IL CARRY



RAL
17

Ruota i contenuti dell'Accumulatore a sinistra di 1 bit attraverso lo stato Carry.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry valga 1. Eseguita l'istruzione:

RAL

l'Accumulatore conterrà $F5_{16}$ e lo stato Carry sarà 0:

Accumulatore	C		Accumulatore	C
01111010	1	⟶	11110101	0

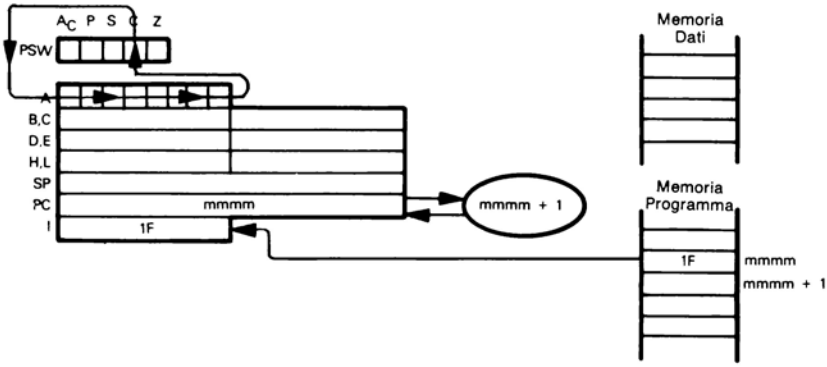
L'istruzione RAL è usata spesso per spostamenti multibyte a sinistra come descritto in "An Introduction To Microcomputers - Volume I". Lo stato Carry è azzerato prima di eseguire il primo spostamento a sinistra; successivamente si trasferiscono i bit di ordine elevato byte nel successivo byte dei bit di ordine basso. Questa è una sequenza di istruzioni che sposta i contenuti di quattro byte di memoria a sinistra di un bit:

```
        LXI   H,DATI   ; Carica l'indirizzo del byte dati di basso ordine
        ANA   A        ; Azzera inizialmente Carry
        MVI   B,3      ; Usa il registro B come contatore
LOOP    MOV   A,M      ; Carica il byte dati nell'Accumulatore
        RAL            ; Ruota a sinistra
        MOV   M,A      ; Rimemorizza il risultato
        INX   H        ; Incrementa l'indirizzo in HL
        DCR   B        ; Decrementa il contatore
        JNZ   LOOP     ; Ritorna per prossimo byte se ce n'è uno
```

CONDIZIONI DI STATO

Si noti che è stata data una particolare attenzione su come vengono influenzati gli stati dalle singole istruzioni. RAL influenza il solo Carry, INX e DCR gli stati Zero, Segno, Parità ma non il Carry che è perciò conservato nel passaggio da una esecuzione RAL alla successiva.

RAR — RUOTA L'ACCUMULATORE A DESTRA ATTRAVERSO IL CARRY



RAR
1F

Ruota a destra i contenuti dell'Accumulatore di un bit attraverso lo stato Carry.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia 1.

Eseguita l'istruzione:

RAR

L'Accumulatore conterrà BD_{16} e lo stato Carry sarà 0:

Accumulatore	C		Accumulatore	C
0 1 1 1 1 0 1 0	1	⟶	1 0 1 1 1 1 0 1	0

L'istruzione RAR è usata spesso per eseguire spostamenti multibyte a destra, come descritto in "An Introduction to Microcomputers - Volume I". Lo stato Carry è azzerato prima dell'esecuzione del primo spostamento a destra; successivamente si trasferiscono i bit di basso ordine di un byte nei bit di ordine elevato del byte successivo. Ecco una sequenza di istruzioni che sposta i contenuti di quattro byte di memoria a destra di un bit.

```
        LXI   H,DATI   ; Carica l'indirizzo del byte di basso ordine
        ANA   A        ; Azzera inizialmente Carry
        MVI   B,3      ; Usa il registro B come contatore
LOOP    MOV   A,M      ; Carica il byte dati nell'Accumulatore
        RAR            ; Ruota a destra
        MOV   M,A      ; Rimemorizza il risultato
        INX   H        ; Incrementa l'indirizzo in HL
        DCR   B        ; Decrementa il contatore
        JNZ   LOOP     ; Ritorna per prossimo byte se ce n'è uno
```

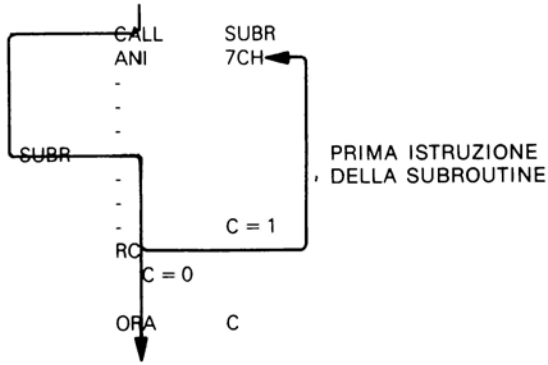
Si veda la descrizione dell'istruzione RAL per una discussione sugli stati.

RC — RITORNA SE LO STATO CARRY E' UGUALE AD 1

RC
D8

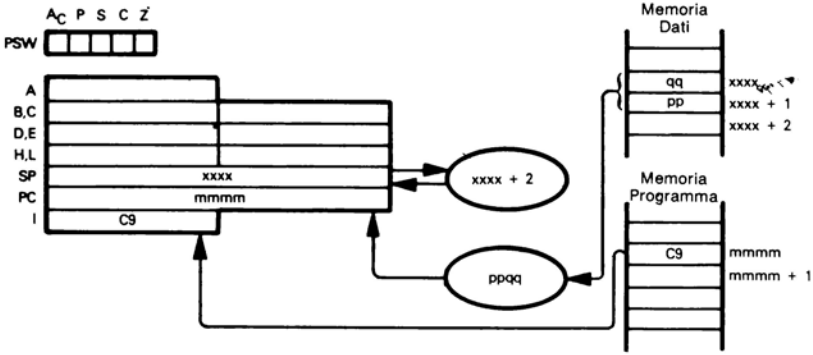
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Carry è diverso da 1, quando viene eseguita l'istruzione RC.

Si consideri la sequenza di istruzioni:



Eseguita l'istruzione RC, se lo stato Carry è uguale ad 1, l'esecuzione ritorna all'istruzione ANI che segue CALL. Se lo stato Carry è uguale a 0 viene eseguita l'istruzione ORA che è la successiva istruzione sequenziale.

RET — RITORNA DA SUBROUTINE



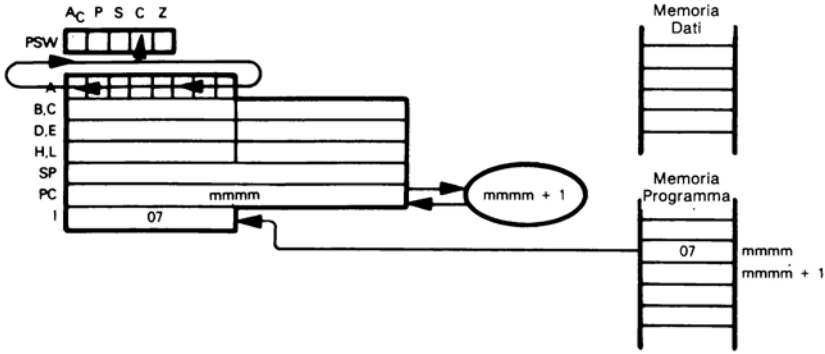
RET
C9

Trasferisce i contenuti dei due byte alla sommità dello Stack al Contatore di Programma; questi due byte forniscono l'indirizzo della successiva istruzione da eseguire. Il precedente contenuto del Contatore di Programma va perso. Viene inoltre incrementato di 2 il Puntatore dello Stack per indirizzare la nuova sommità dello stack.

Ogni subroutine deve contenere almeno una istruzione di Ritorno (o Ritorno Condizionale); questa è l'ultima istruzione eseguita all'interno della subroutine ed origina l'esecuzione del ritorno al programma chiamante.

Per una dettagliata descrizione dell'esecuzione dell'istruzione RET, si veda il Capitolo 5.

RLC — RUOTA L'ACCUMULATORE A SINISTRA



RLC
07

Ruota i contenuti dell'Accumulatore a sinistra di un bit.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia ad 1.

Dopo che è stata eseguita l'istruzione:

RLC

l'Accumulatore conterrà $F4_{16}$ e lo stato Carry sarà al livello logico 0:

Accumulatore	C		Accumulatore	C
0 1 1 1 1 0 1 0	1	⟶	1 1 1 1 0 1 0 0	0

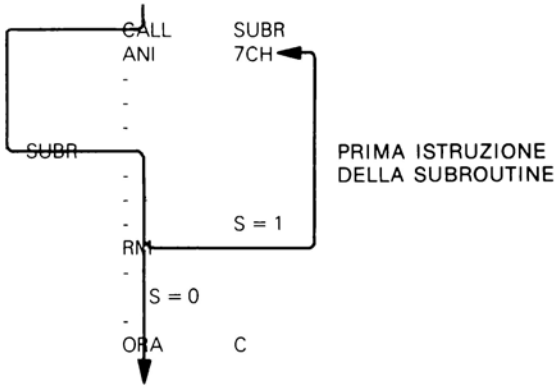
RLC potrebbe essere usata come istruzione logica.

RM — RITORNA SE LO STATO SIGN E' UGUALE AD 1

RM
F8

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Sign è 0, quando viene eseguita l'istruzione RM.

Si consideri la sequenza di istruzioni:



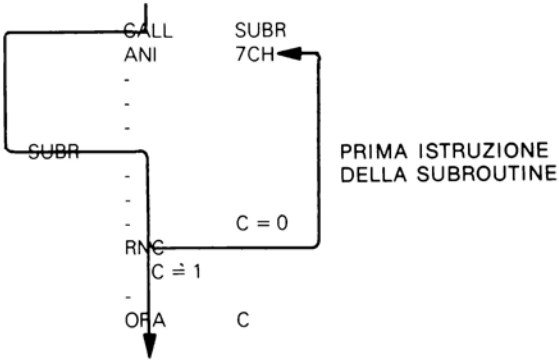
Dopo che è stata eseguita l'istruzione RM, se lo stato è uguale a 1, l'esecuzione ritorna all'istruzione ANI, che esegue la CALL. Se lo stato Sign è uguale a 0 viene eseguita l'istruzione ORA che è la successiva in ordine sequenziale.

RNC — RITORNA SE LO STATO CARRY E' UGUALE A 0

RNC
D0

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Carry è uguale ad 1 quando viene eseguita l'istruzione RNC.

Si consideri la sequenza di istruzioni:



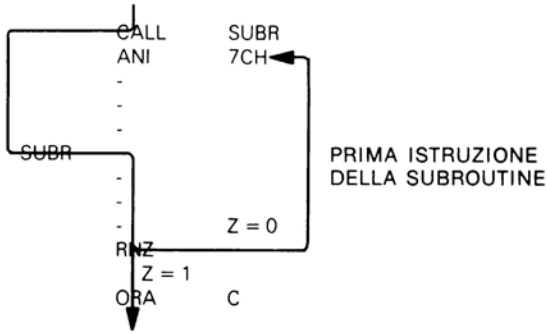
Dopo che è stata eseguita l'istruzione RNC, se lo stato Carry è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Carry è uguale ad 1 viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RNZ — RITORNA SE LO STATO ZERO E' UGUALE A 0

RNZ
C0

Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Zero è uguale ad 1, quando viene eseguita l'istruzione RNZ.

Si consideri la sequenza di istruzioni:



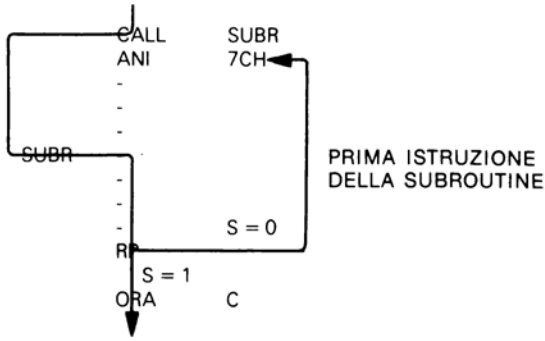
Dopo che e stata eseguita l'istruzione RNZ, se lo stato Zero è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Zero è uguale ad 1, viene eseguita l'istruzione ORA che è la successiva in ordine sequenziale.

RP — RITORNA SE LO STATO SIGN E' UGUALE A 0

RP
F0

Questa è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Sign è uguale ad 1, durante l'esecuzione dell'istruzione RP.

Si consideri la sequenza di istruzioni:



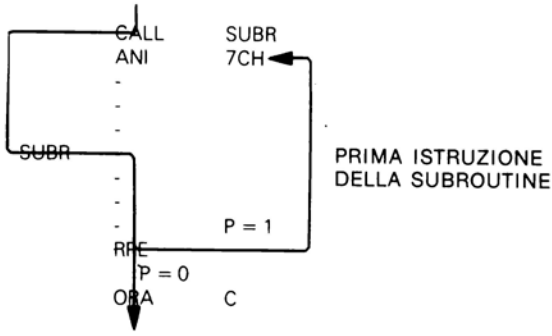
Dopo che è stata eseguita l'istruzione RP, se lo stato Sign è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Sign è uguale ad 1, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RPE — RITORNA SE LO STATO PARITA' E' UGUALE AD 1

RPE
E8

Questa istruzione è identica alla RET, tranne il fatto che il ritorno non viene eseguito se lo stato Parita è uguale a 0, quando viene eseguita l'istruzione RPE.

Si consideri la sequenza di istruzioni:



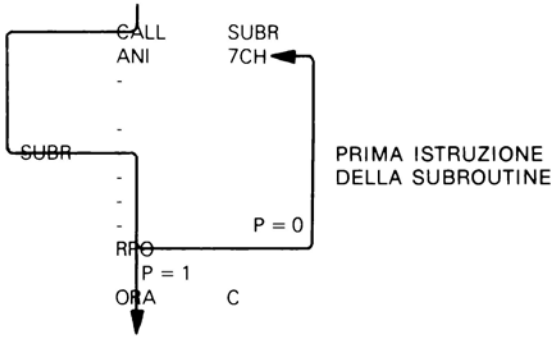
Dopo che è stata eseguita l'istruzione RPE, se lo stato Parità è uguale ad 1, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Parità è uguale a 0, viene eseguita l'istruzione ORA che è la successiva in ordine sequenziale.

RPO — RITORNA SE LO STATO PARITA' E' UGUALE A 0

RPO
E0

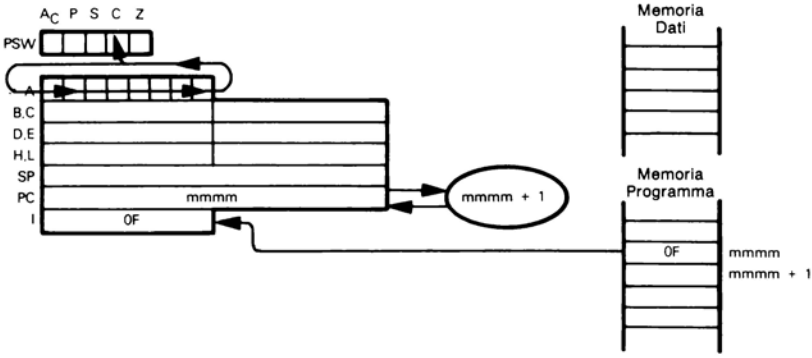
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Parità è uguale ad 1, durante l'esecuzione della RPE.

Si consideri la sequenza di istruzioni:



Dopo che è stata eseguita l'istruzione RPO, se lo stato Parità è uguale a 0, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Parità è uguale ad 1, viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

RRC — RUOTA L'ACCUMULATORE A DESTRA



RRC
0F

Ruota a destra di un bit i contenuti dell'Accumulatore.

Si supponga che l'Accumulatore contenga $7A_{16}$ e lo stato Carry sia ad 1.

Dopo l'esecuzione dell'istruzione:

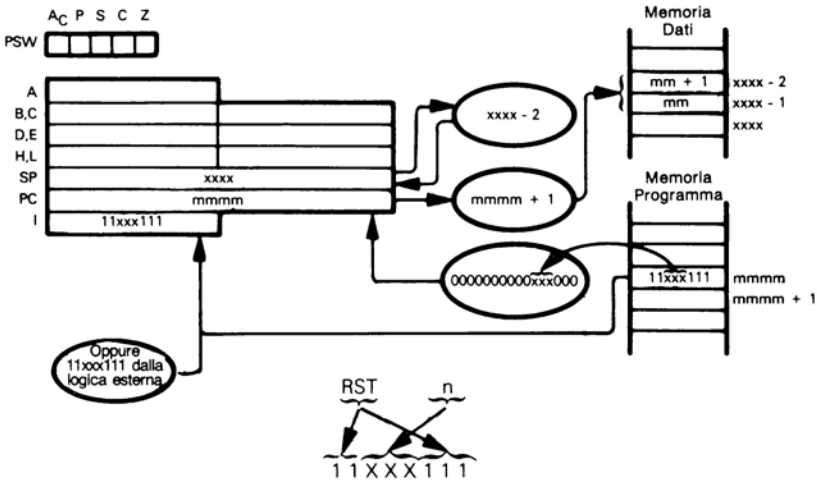
RRC

l'Accumulatore conterrà $3D_{16}$ e lo stato Carry sarà 0:

Accumulatore	C		Accumulatore	C
0 1 1 1 1 0 1 0	1	⟶	0 0 1 1 1 1 0 1	0

RRC potrebbe essere usata come istruzione logica.

RST — RESTART (RIPARTI)



Chiama la subroutine iniziante all'indirizzo di memoria specificato con N.

Quando viene eseguita l'istruzione:

RST 3

viene richiamata la subroutine iniziante alla locazione di memoria 0018_{16}.

Il precedente contenuto del Contatore di Programma viene mandato alla sommità dello stack.

Normalmente l'istruzione RST viene impiegata assieme all'esecuzione di interrupt, come sarà descritto nel Capitolo 5.

CHIAMATA DI SUBROUTINE USANDO RST

Se si hanno applicazioni che non fanno uso di tutti i codici istruzione RST per il servizio di interrupt non si apprezza la possibilità di chiamta di subroutine impiegando l'istruzione RST. Iniziando le subroutine ad appropriati indirizzi RST in fase di chiamata di queste subroutine è sufficiente un'istruzione a singolo byte RST al posto della istruzione CALL di tre byte.

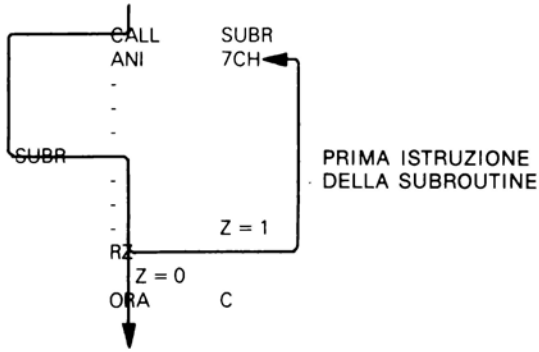
RZ — RITORNA SE LO STATO ZERO E' UGUALE AD 1

RZ

C8

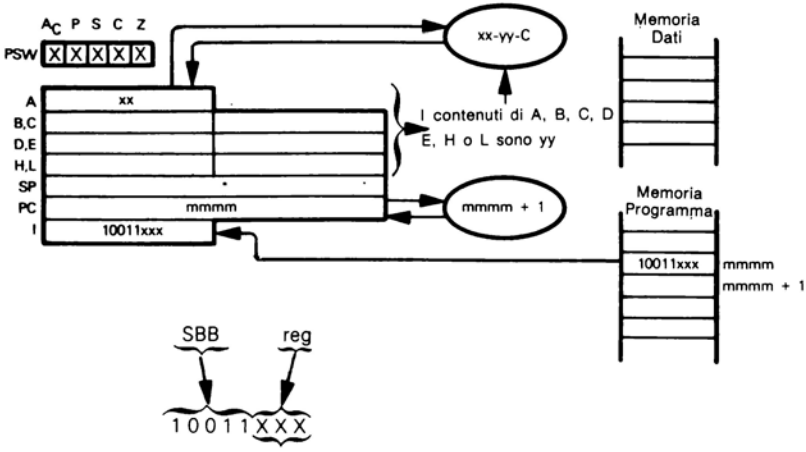
Questa istruzione è identica alla RET tranne il fatto che il ritorno non viene eseguito se lo stato Zero è uguale a 0 quando si esegue l'istruzione RZ.

Si consideri la sequenza di istruzioni:



Dopo che è stata eseguita l'istruzione RZ, se lo stato Zero è uguale ad 1, l'esecuzione ritorna all'istruzione ANI, che segue la CALL. Se lo stato Zero è uguale a 0 viene eseguita l'istruzione ORA, che è la successiva in ordine sequenziale.

SBB — SOTTRAI UN REGISTRO O MEMORIA DALL'ACCUMULATORE CON PRESTITO



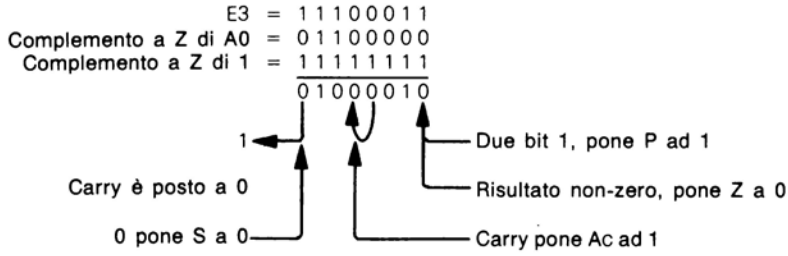
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae i contenuti del registro specificato e lo stato Carry dall'Accumulatore, trattando i contenuti dei registri come semplici dati binari.

Si supponga xx = $E3_{16}$, E contenga $A0_{16}$, C = 1. Dopo l'esecuzione dell'istruzione:

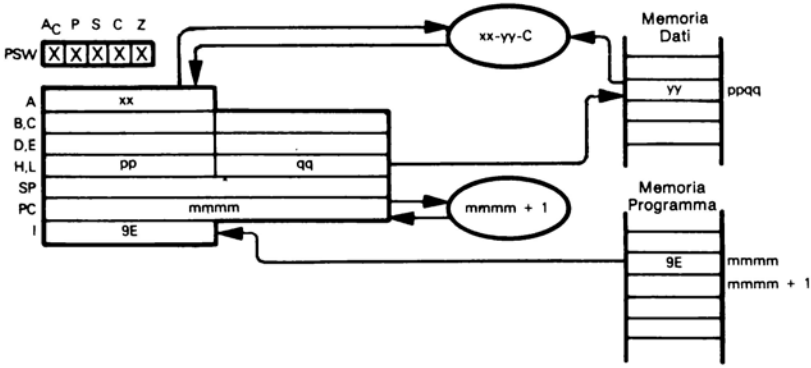
SBB E

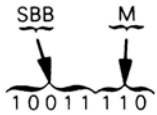
l'Accumulatore conterrà 42_{16}.



Si noti che il Carry risultante è complementato.

Anche i contenuti di un byte di memoria possono essere sottratti dall'Accumulatore:





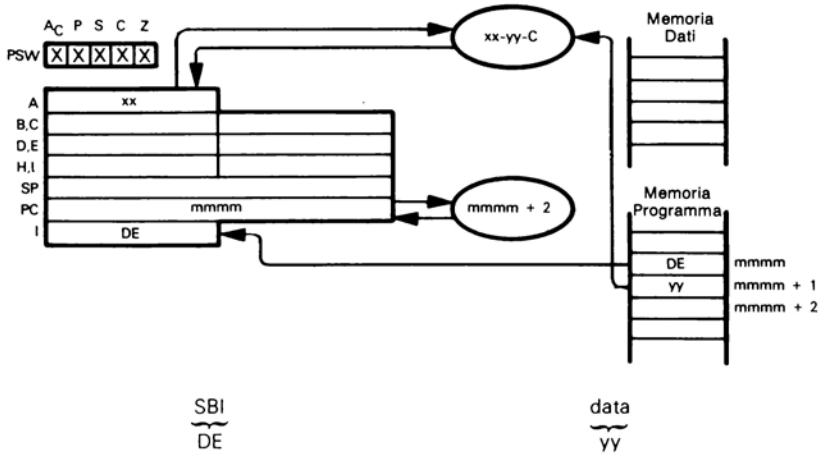
Se xx = $E3_{16}$, yy = $A0_{16}$ e C = 1 allora l'esecuzione dell'istruzione:

SBB M

genera lo stesso risultato dell'esecuzione dell'istruzione SBB E, appena descritta.

L'istruzione SBB nella sottrazione multibyte dopo che i bit di basso ordine sono stati elaborati usando l'istruzione SUB.

SBI — SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE CON PRESTITO

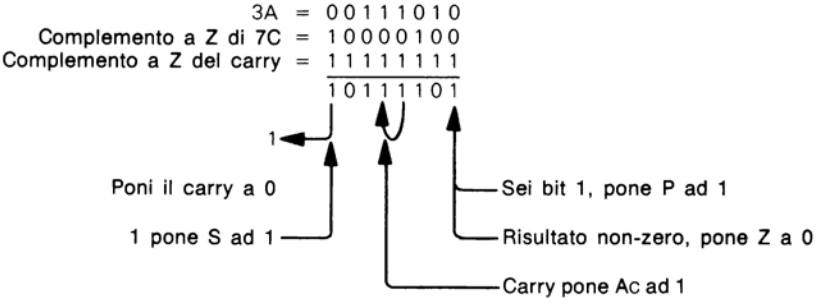


Sottrae i contenuti del secondo byte di istruzione in codice e lo stato Carry dall'Accumulatore.

Si supponga xx = $3A_{16}$ e lo stato Carry sia uguale ad 1. Dopo che l'istruzione:

SBI 7CH

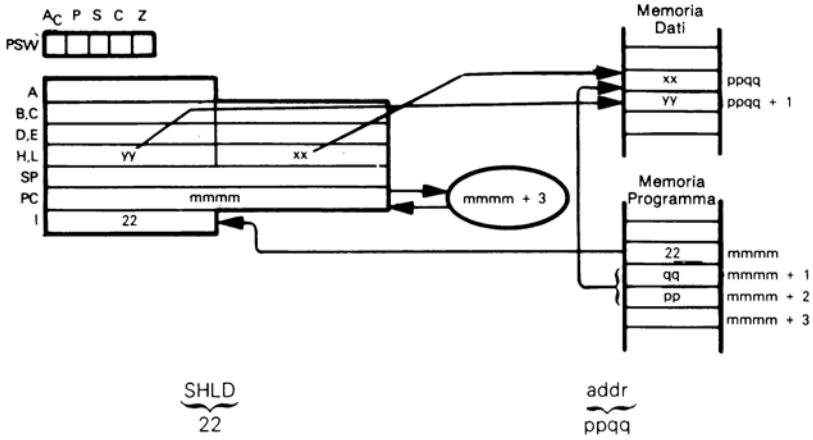
è stata eseguita, l'Accumulatore conterrà BD_{16}:



Si noti che il Carry risultante è complementato.

Questa istruzione non viene usata comunemente come SUI.

SHLD — MEMORIZZA DIRETTO I REGISTRI H ED L



Il secondo e terzo byte in codice oggetto forniscono l'indirizzo di memoria del byte dati nel quale vengono scritti i contenuti del registro L. I contenuti del registro H vengono scritti nel byte dati sequenzialmente successivo.

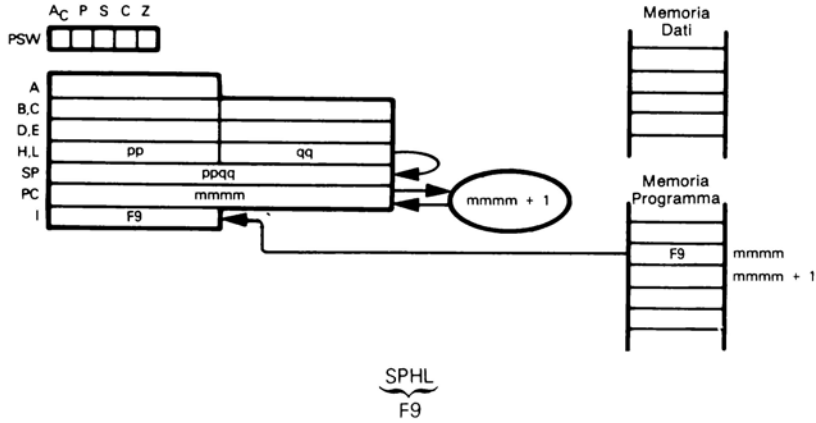
Si suppoga xx = $2C_{16}$ ed yy = $3A_{16}$. Dopo che è stata eseguita l'istruzione:

```
LABEL   EQU     084AH
        -
        -
        -
        SHLD    LABEL
```

il byte di memoria $084A_{16}$ conterrà $2C_{16}$ ed il byte di memoria $084B_{16}$ conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assemblatore e non un'istruzione, esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ quando appare LABEL.

SPHL — CARICA IL PUNTATORE DELLO STACK DAI REGISTRI H E L



Muove i contenuti dei registri H ed L al Puntatore dello Stack.

Si supponga che pp = 08_{16} e qq = $3F_{16}$. Dopo che è stata eseguita l'istruzione:

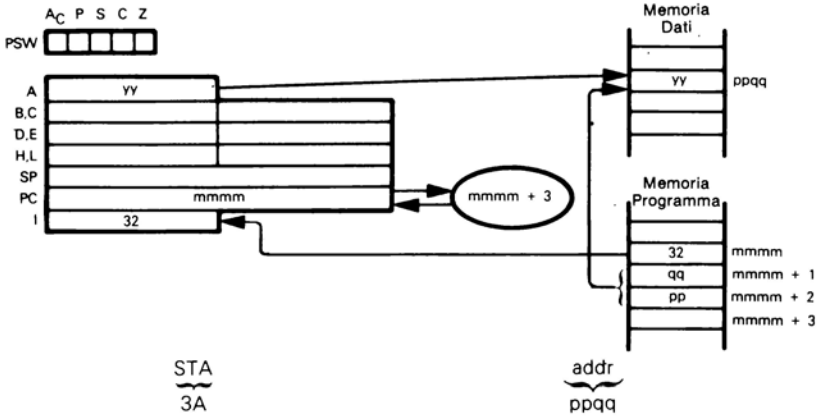
SPHL

il Puntatore dello Stack conterrà $083F_{16}$.

L'istruzione SPHL può essere usata per accedere a due stacks — con un indirizzo conservato nei reigstri H ed L. Gli Stacks sono spesso usati in questo modo per accedere alle stringhe di testo o dati qualsiasi ai cui byte si deve accedere serialmente.

Il punto importante da ricordare è che questa logica dello stack può essere usata al posto dell'indirizzamento implicato di memoria con auto-incremento.

STA — IMMAGAZZINA L'ACCUMULATORE IN MEMORIA USANDO L'INDIRIZZAMENTO DIRETTO



Memorizza i contenuti dell'Accumulatore del byte dell'istruzione STA in codice oggetto.

Si supponga che l'Accumulatore contenga $3A_{16}$. Dopo l'esecuzione dell'istruzione:

```
LABEL  EQU    084AH
        —
        —
        —
       STA    LABEL
```

il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

Si ricordi che EQU è un ordine assemblatore e non una istruzione; esso dice all'Assemblatore di usare il valore a 16 bit $084A_{16}$ dovunque appare LABEL.

L'istruzione:

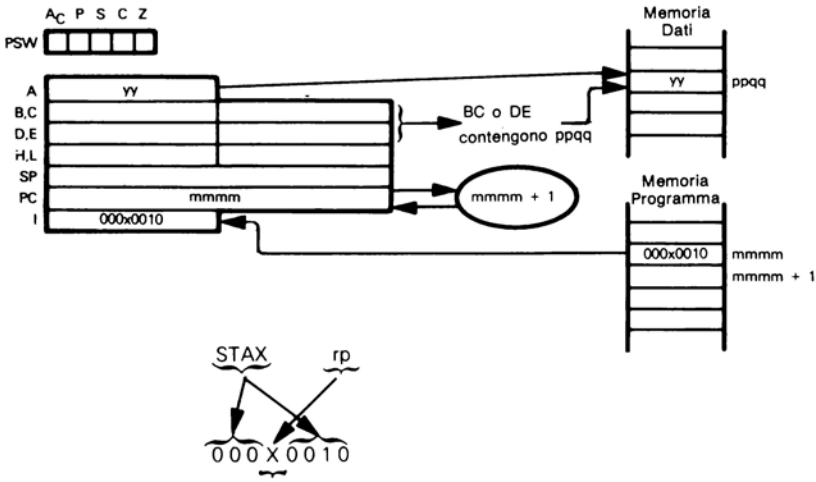
```
       STA    LABEL
```

è equivalente alle due istruzioni:

```
       LXI    H,LABEL
       MOV    M,A
```

Quando si sta immagazzinando i valori di singoli dati in memoria è conveniente usare l'istruzione STA; essa usa un'istruzione e tre byte del programma oggetto per fare lo stesso che la combinazione LXI MOV fa in due istruzioni e quattro byte del programma oggetto. Inoltre la combinazione LXI MOV usa i registri H ed L, l'istruzione STA invece no.

STAX – IMMAGAZZINA IL CONTENUTO DELL'ACCUMULATORE NELLA LOCAZIONE DI MEMORIA INDIRIZZATA DA UNA COPPIA DI REGISTRI



0 se rp è B, rappresentante BC
1 se rp è D, rappresentante DE

Immagazzina i contenuti dell'Accumulatore nel byte di memoria indirizzato dalla coppia di registri BC o DE.

Si supponga che il registro B contenga 08_{16}, il registro C contenga $4A_{16}$ e l'Accumulatore contenga $3A_{16}$. Dopo che è stata eseguita l'istruzione:

STAX B

il byte di memoria $084A_{16}$ conterrà $3A_{16}$.

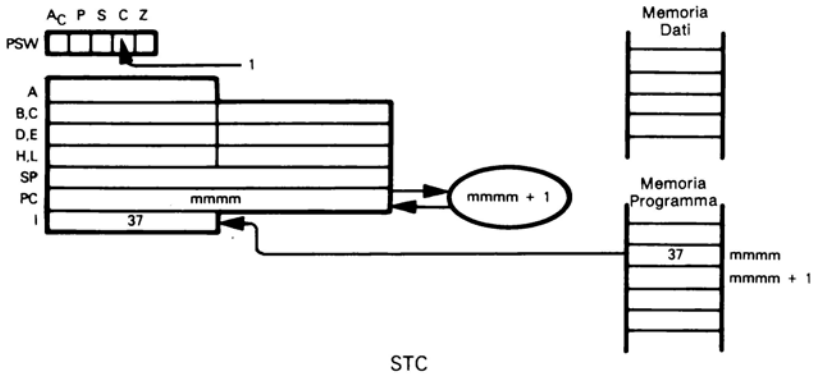
Si noti che non esiste l'istruzione STAX H perchè è identica all'istruzione MOV M,A.

Normalmente le istruzioni STAX ed LXI saranno usate assieme perchè l'istruzione LXI carica un indirizzo a 16 bit nei registri BC o DE, come segue:

```
LXI     B,084AH
STAX    B
```

Si noti che l'istruzione STAX immagazzinerà dati solo dall'Accumulatore, mentre l'istruzione MOV immagazzinerà dati da qualunque registro.

STC — PONI AD 1 LO STATO CARRY

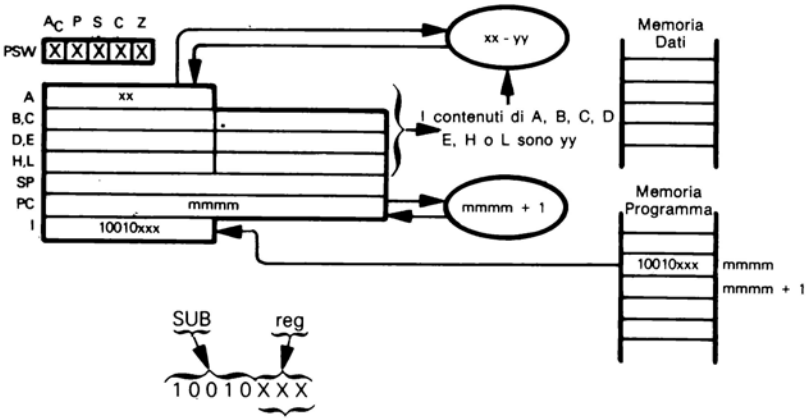


STC
37

Quando viene eseguita l'istruzione STC lo stato Carry è posto ad 1, indipendentemente dal suo precedente valore. Nessun altro stato o registro viene influenzato da questa istruzione.

SUB — SOTTRAI IL CONTENUTO DI UN REGISTRO O MEMORIA DALL'ACCUMULATORE

Questa istruzione assume due forme. La prima sottrae i contenuti di un registro dall'Accumulatore.



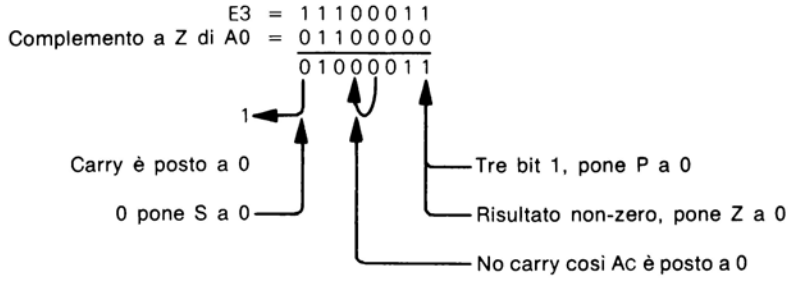
000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Sottrae i contenuti del registro specificato dall'Accumulatore, trattando i contenuti del registro come semplici dati binari.

Si supponga xx = $E3_{16}$ ed il registro E contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione:

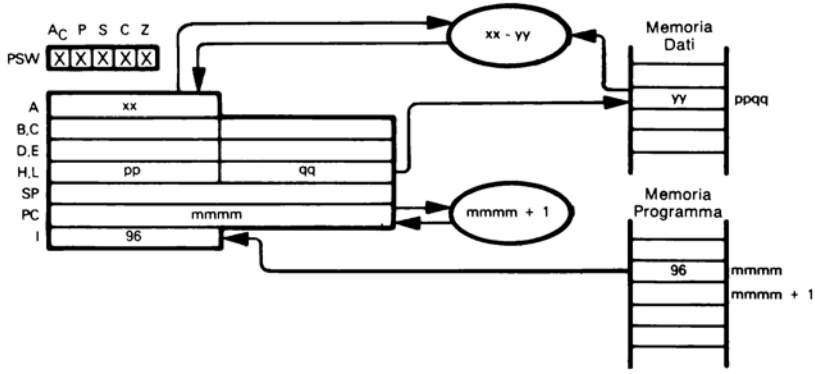
SUB E

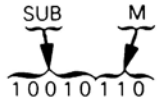
l'Accumulatore conterrà 43_{16}:



Si noti che il Carry risultante è complementato.

Anche i contenuti dei byte di memoria essere sottratti dall'Accumulatore:





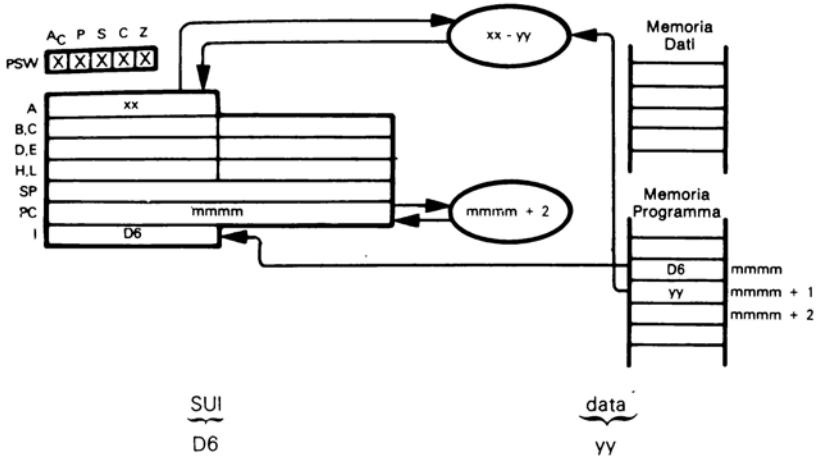
Se xx = $E3_{16}$ ed yy = $A0_{16}$ allora l'esecuzione dell'istruzione:

SUB M

genera lo stesso risultato dell'esecuzione dell'istruzione SUB E, appena descritta.

L'istruzione SUB è usata per eseguire sottrazioni a byte singolo, o per i byte di basso ordine di sottrazioni multibyte.

SUI — SOTTRAI IMMEDIATO DATI DALL'ACCUMULATORE

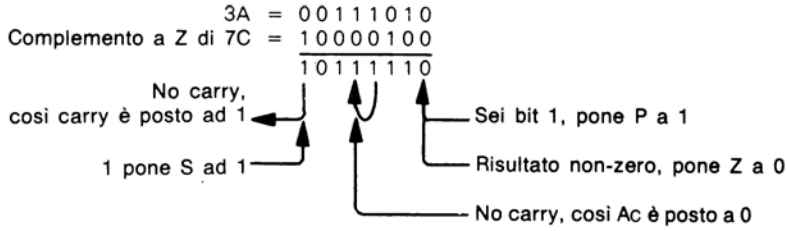


Sottrae i contenuti del secondo byte del codice di istruzione dall'Accumulatore.

Si supponga xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione:

SBI 7CH

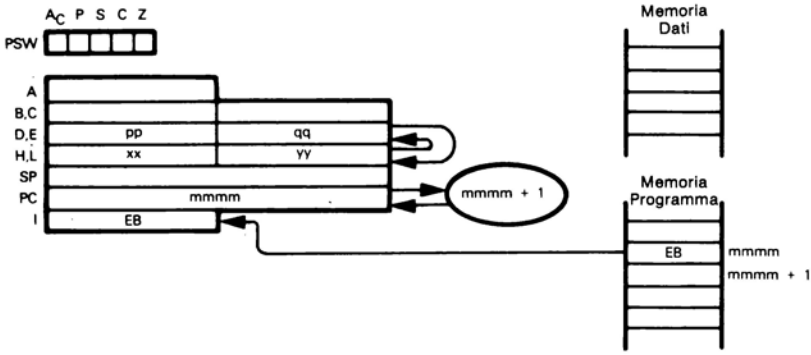
è stata eseguita, l'Accumulatore conterrà BE_{16}:



Si noti che il Carry risultante è complementato.

Questa istruzione è lìdeale per la sottrazione immediata.

XCHG — SCAMBIA I CONTENUTI DEI REGISTRI DE ED HL



XCHG

EB

I contenuti dei registri D ed E sono scambiati con quelli dei registri H ed L.

Si supponga pp = 03_{16}, qq = $2A_{16}$, xx = 41_{16} ed yy = FC_{16}. Dopo l'esecuzione dell'istruzione:

XCHG

H conterrà 03_{16}, L conterrà $2A_{16}$, D conterrà 41_{16} ed E conterrà FC_{16}.

Le due istruzioni:

```
XCHG
MOV     A,M
```

sono equivalenti a:

```
LDAX    D
```

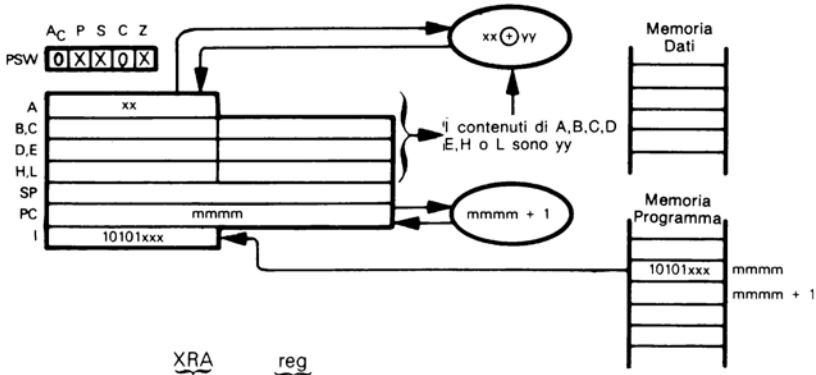
Però se si vuole caricare l'indirizzo dati mediante i registri D ed E nel registro B, le sue istruzioni:

```
XCHG
MOV     B,M
```

non hanno la singola istruzione equivalente.

XRA — OR ESCLUSIVO DI UN REGISTRO O MEMORIA CON L'ACCUMULATORE

Questa istruzione assume due forme. La prima opera l'OR esclusivo dei contenuti di un registro con l'Accumulatore:



000 per reg = B
001 per reg = C
010 per reg = D
011 per reg = E
100 per reg = H
101 per reg = L
111 per reg = A

Opera l'OR esclusivo dei contenuti del registro specificato con l'Accumulatore, trattando i contenuti dei registri come semplici dati binari.

Si supponga xx = $3E_{16}$ ed il registro E contenga $A0_{16}$. Dopo l'esecuzione dell'istruzione:

XRA E

l'Accumulatore conterrà 43_{16}:

```
E3 = 11100011
A0 = 10100000
     --------
     01000011
```

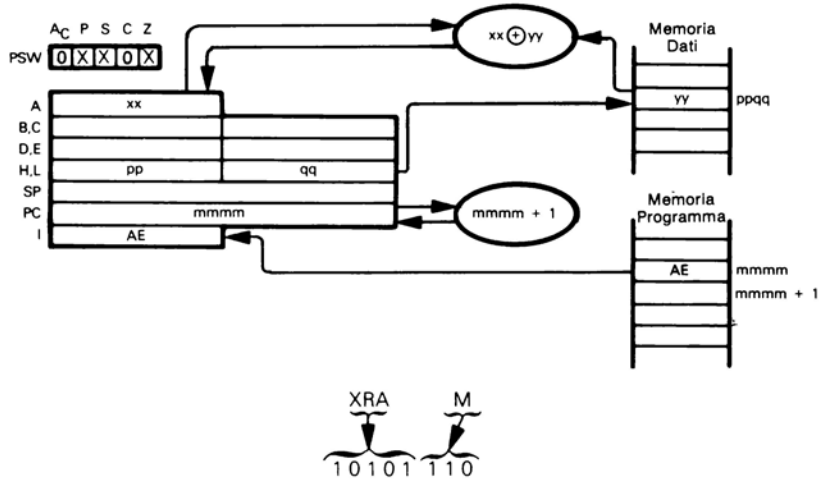
Carry è posto a 0

0 pone S a 0

Tre bit 1, pone P a 0

Risultato non-zero, pone Z a 0

Si può operare l'OR esclusivo anche tra i contenuti di un byte di memoria e l'Accumulatore.



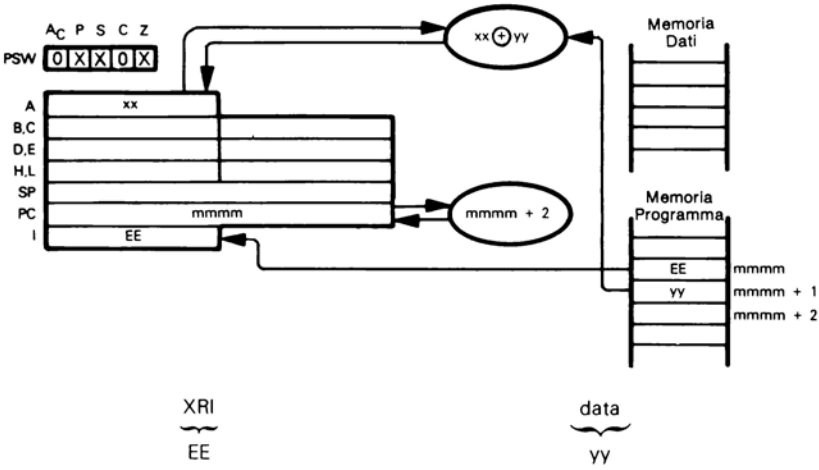
Se xx = $E3_{16}$ ed yy = $A0_{16}$ allora l'esecuzione dell'istruzione:

XRA M

genera lo stesso risultato dell'esecuzione dell'istruzione XRA E, appena descritta.

L'istruzione di OR esclusivo è usata come test per cambiare i bit di stato.

XRI – OR ESCLUSIVO IMMEDIATO DI DATI CON L'ACCUMULATORE

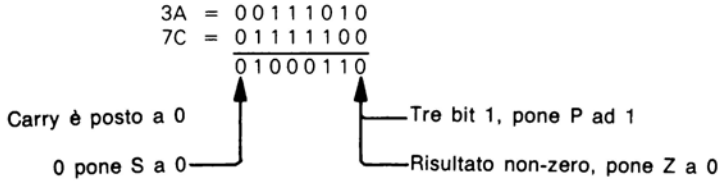


Opera l'OR esclusivo dei contenuti del secondo byte del codice di istruzione con l'Accumulatore.

Si supponga xx = $3A_{16}$. Dopo l'esecuzione dell'istruzione:

XRI 7CH

l'Accumulatore conterrà 46_{16}:

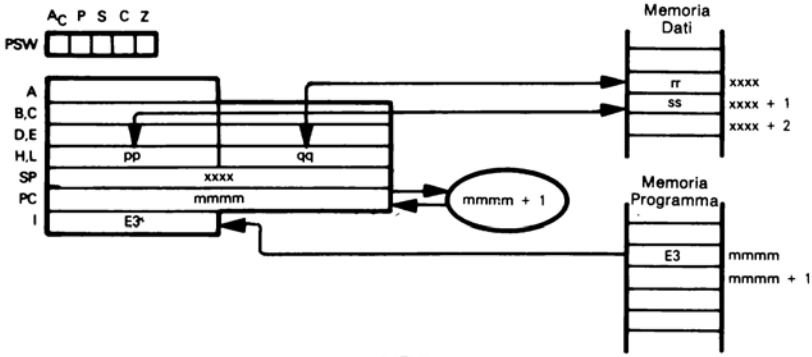


Questa è una istruzione logica di routine; viene impiegata spesso per la complementazione di bit. Per esempio l'istruzione:

XRI 03H

complementerà incondizionatamente i due bit di basso ordine dell'Accumulatore.

XTHL — SCAMBIA LA SOMMITA' DELLO STACK CON HL



XTHL
E3

Scambia i contenuti del registro H col byte alla sommità dello stack e quelli del registro H col byte sotto la sommità dello stack.

Si supponga pp = 21_{16}, qq = FA_{16}, rr = $3A_{16}$, ss = $E2_{16}$. Dopo l'esecuzione della istruzione:

XTHL

H conterrà $E2_{16}$, L conterrà $3A_{16}$ ed i due byte alla sommità dello stack conterranno rispettivamente FA_{16} e 21_{16}.

Le due istruzioni:

XTHL
XTHL

eseguite in sequenza si neutralizzano.

L'istruzione XTHL è usata per accedere e manipolare dati alla sommità dello stack come illustrato nella discussione sulle subroutine multiple del Capitolo 5.

Capitolo 7

ALCUNE SUBROUTINE IMPIEGATE COMUNEMENTE

Esiste un certo numero di operazioni che possono ricorrere in molti programmi al microcalcolatore, indipendentemente dall'applicazione. Questo capitolo fornirà un certo numero di sequenze d'istruzioni usate spesso.

Per rendere più efficiente l'utilizzazione di questo capitolo si potrebbe studiare ogni subroutine in modo da conoscerla a sufficienza per modificarla. Come semplice esercizio si potrebbe tentare di riscrivere la subroutine in modo che essa esegua lo stesso compito impiegando meno cicli di esecuzione o meno istruzioni od entrambi. Successivamente si riscriva il programma per realizzare variazioni. Per esempio viene illustrata la moltiplicazione binaria a 16-bit, come costruire una routine per moltiplicare numeri di 32 bit? Si consideri ogni esempio come una sequenza d'istruzioni tipica, illustrativa che si modificherà facilmente per soddisfare le richieste immediate.

I semplici programmi al livello coperto da questo capitolo rientrano in una delle seguenti categorie:

1) **Indirizzamento di memoria**
2) **Movimento dati**
3) **Aritmetica**
4) **Sequenza logica di esecuzione del programma**

Si descriveranno programmi nella precedente sequenza di categoria.

INDIRIZZAMENTO DELLA MEMORIA

Benchè i modi dell'indirizzamento di memoria dell'8080/9080 siano limitati all'indirizzamento diretto ed implicato, semplici sequenze d'istruzione consentono la simulazione di qualsiasi altro modo di indirizzamento.

Si mostrerà l'incremento ed il decremento automatici, l'indirizzamento con indice, l'indirizzamento diretto e l'indirizzamento indiretto con post-indicizzazione; tutti questi modi di indirizzamento sono descritti ed illustrati in "An Introduction to Microcomputers" Volume I.

AUTO-INCREMENTO ED AUTO-DECREMENTO

Una debolezza del set di istruzione dell'8080/9080, rispetto agli altri microcalcolatori, **è la mancanza della possibilità di auto-incremento ed auto-decremento implicato;** le routine per muovere dati descritte nel seguito di questo capitolo illustrano la richiesta gratuita di incrementare/decrementare costantemente gli indirizzi durante la manipolazione di dati nei buffer — o qualsiasi blocco di byte della memoria contigui.

INDIRIZZAMENTO DI MEMORIA MEDIANTE IL PUNTATORE DELLO STACK

In questi casi si può impiegare il Puntatore dello Stack per realizzare l'indirizzamento di memoria implicato con auto-incremento ed auto-decremento. Comunque si deve partire con qualche restrizione di programma:

1) Si deve impiegare l'istruzione Push in luogo di una scrittura-in-memoria e l'istruzione Pop in luogo di

una lettera-da-memoria. Questo limita ad auto-decrementare durante la lettura e ad auto-incrementare durante la lettura.

2) Nella memoria si accede per coppie di byte; si ricordi che le istruzioni Push e Pop trattano con coppie di registri, non con unità dati a singolo byte.
   Questo può essere un vantaggio se taglia a metà il numero di istruzioni eseguite; è uno svantaggio se si sta trattando con buffers di lunghezza di byte dispari o se, per qualunque ragione, non si può manipolare i dati in unità a 16-bit.
3) I contenuti precedenti del Puntatore dello Stack indirizzano la sommità dello stack effettiva in modo che questa venga conservata mentre si impiega il Puntatore dello Stack come un registro di indirizzo di memoria. Questo naturalmente significa che non si possono impiegare subroutines od interrupts, oppure accessi allo stack, finchè non si e ri-immagazzinato il Puntatore dello Stack.

CONSERVAZIONE DEL PUNTATORE DELLO STACK

Perchè si conservano i contenuti precedenti del Puntatore dello Stack? Il set d'istruzione dell'8080 fornisce la istruzione SPHL per caricare il Puntatore dello Stack dai registri H ed L; **ma non esiste un'istruzione che muove i contenuti del Puntatore dello Stack nei registri HL -- o qualunque altro registro. Invece si devono azzerare i registri H ed L e quindi eseguire un'istruzione DAD come segue:**

```
MVI     H,0     ; Azzera H
MOV     L,H     ; Azzera L
DAD     SP      ; Muove SP ad HL
```

Si noti che l'istruzione DAD modificherà lo stato Carry. Ora si possono conservare i contenuti del Puntatore dello Stack in un'altra coppia di registri:

```
XCHG            ; Conserva HL in DE
```

oppure:

```
MOV     B,H     ; Conserva H L in B C
MOV     C,L
```

Oppure si possono conservare due byte della memoria di lettura-scrittura per l'immagazzinamento temporaneo del Puntatore dello Stack:

```
SHLD    STAK    ; Conserva HL in STACK e STAK+1
```

RI-IMMAGAZZINAMENTO DEL PUNTATORE DELLO STACK

Il ri-immagazzinamento dei contenuti conservati del Puntatore dello Stack è molto facile; da BC queste istruzioni applicano:

```
MOV     H,B     ; Muove BC ad HL
MOV     L,C
SPHL            ; Muove HL ad SP
```

Da DE queste istruzioni applicano:

```
XCHG            ; Muove DE ad HL
SPHL            ; Muove HL ad SP
```

Dalla memoria queste istruzioni applicano:

```
LHLD    STAK    ; Carica HL da STAK e STAK+1
SPHL            ; Muove HL ad SP
```

CARICAMENTO DI INDIRIZZO NEL PUNTATORE DELLO STAK

Una volta che i contenuti del Puntatore dello Stack sono stati conservati si può caricare un nuovo indirizzo nel Puntatore dello Stack, in modo immediato:

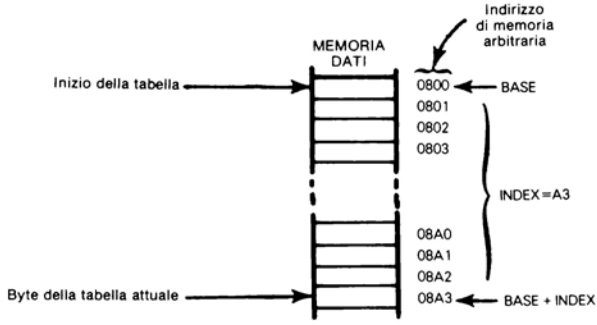
```
LXI     SP,ADDR
```

Oppure si può caricare un indirizzo che può essere conservato in due byte della memoria di lettura/scrittura:

```
LHLD    ADDR    ; Carica l'indirizzo in HL
SPHL            ; Muove HL ad SP
```

INDIRIZZAMENTO CON INDICE

L'indirizzamento con indice richiede che l'indirizzo sia calcolato come un indirizzo base più lo spostamento, fornito da un registro con indice; ecco un esempio:



Le variazioni sono determinate dal fatto che la BASE è un indirizzo invariante, mentre l'INDICE varia continuamente; si accederà perciò alla BASE come un valore a 16-bit immediato, ma l'INDICE è assegnato a due byte della memoria di lettura/scrittura, con indirizzi INDX ed INDX+1. Ora si può originare un indirizzo con indice come segue:

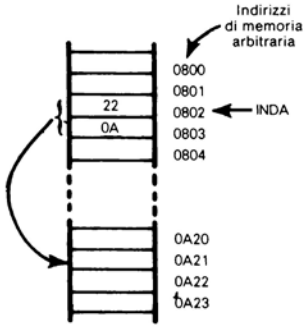
```
LXI   B,BASE   ; Carica l'indirizzo base in BC
LHLD  INDX     ; Carica l'indice in HL
DAD   B        ; Somma BC ad HL
```

Qual'è la variazione del valore dell'indice attuale? Una volta costruito si può ri-immagazzinare il nuovo valore dell'indice sottraendo la BASE dei contenuti di HL, come segue.

```
LXI   B,BASE   ; Carica l'indirizzo base in BC
MOV   A,L      ; Sottrae C da L
SUB   C
STA   INDX     ; Conserva in INDX
MOV   A,H      ; Sottrae B da H con prestito
SBB   B
STA   INDX+1   ; Conserva in INDX+1
```

INDIRIZZAMENTO INDIRETTO

L'indirizzamento indiretto specifica che l'indirizzo di memoria richiesto è immagazzinato in due byte di memoria:



Nella precedente illustrazione i byte di memoria 0802_{16} e 0803_{16} contengono gli indirizzi di memoria richiesti: $0A22_{16}$. Con riferimento all'8080 si ha la manipolazione di indirizzi a 16-bit, il byte di indirizzo di basso ordine è mostrato precedendo il byte di indirizzo di ordine elevato.

Queste istruzioni simulano l'indirizzo indiretto:

```
        LHLD    IND A   ; Carica l'indirizzo in HL
; Ora si accede alla memoria impiegando l'indirizzamento normale,
; Implicato, con HL che fornisce l'indirizzo
```

INDIRIZZAMENTO INDIRETTO, A POST-INDICE

Si osservi ancora la tabella cui si accede impiegando l'indirizzamento con indice. Si supponga che l'indirizzo base della tabella (BASE) si trovi in due byte di memoria, indirizzati da INDA ed INDA+1. Si può completare l'indirizzo attuale della tabella come segue:

```
        LHLD    INDA    ; Carica l'indirizzo base in HL
        MOV     B,H     ; Conserva in B,C
        MOV     C,L
        LHLD    INDX    ; Carica l'indice in HL
        DAD     B       ; Somma BC ad HL
```

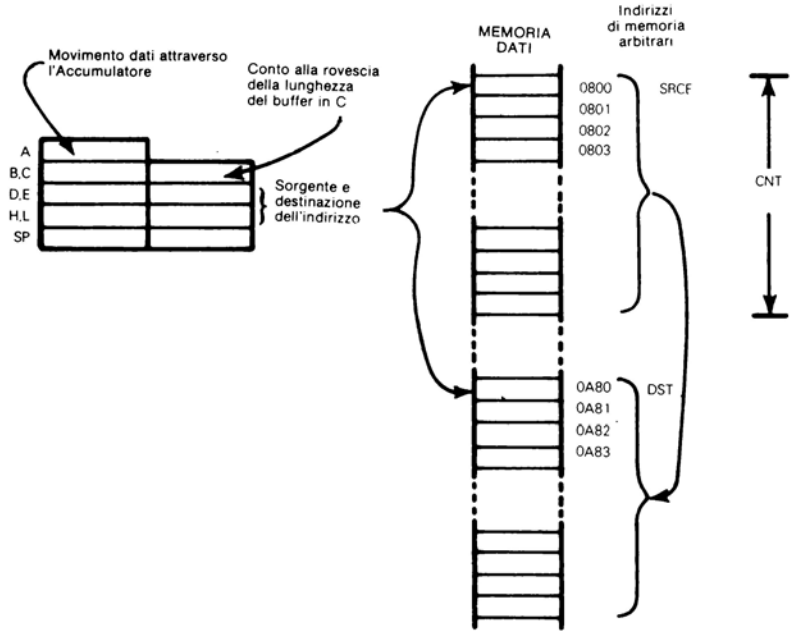
Questo è equivalente all'indirizzamento indiretto, a post-indice.

MOVIMENTO DATI

Si esaminerà ora qualche sequenza d'istruzione che localizza e muove blocchi contigui di byte dati-buffer di dati di qualsiasi lunghezza.

MOVIMENTO DI BLOCCHI SEMPLICI DI DATI

Cominciando con un programma molto semplice, si consideri il movimento dei contenuti di un blocco contiguo di byte della memoria da un'area di memoria ad un'altra. Il modo più semplice di esecuzione di questa operazione è di indirizzare la sorgente e destinazione dei buffers impiegati i registri DE ed HL. La seguente mappa di memoria illustra l'operazione di movimento dati:



Questo è il programma di movimento dati:

```
      LXI   H,SRCE   ; Carica l'indirizzo sorgente in HL
      LXI   D,DST    ; Carica l'indirizzo di destinazione in DE
      MVI   C,CNT    ; Carica il conteggio di byte in C
LOOP  MOV   A,M      ; Carica il byte sorgente
      INX   H        ; Incrementa l'indirizzo sorgente
      STAX  D        ; Immagazzina nella destinazione
      INX   D        ; Incrementa l'indirizzo di destinazione
      DCR   C        ; Decrementa la lunghezza del buffer
      JNZ   LOOP     ; Ritorna se il buffer non è vuoto.
```

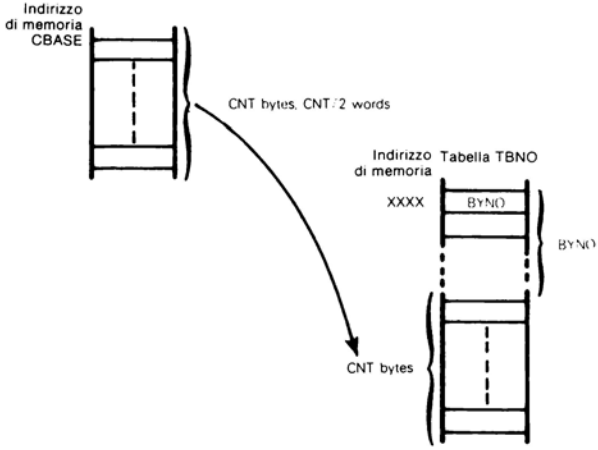
CONSULTAZIONE DI TABELLA MULTIPLA

Si consideri ora una consultazione di tabella multipla. Questa è una variazione più complessa del movimento dati che è stato appena descritto.

Un numero indefinito di tabelle dati ha il proprio indirizzo di partenza immagazzinato in una tabella con indice. L'indirizzo di partenza della tabella con indice è dato dalla label TABX:

Un certo numero di byte dati sono immagazzinati temporaneamente, a partire da una locazione di memoria identificata dalla label C BASE. Il numero corrente dei bytes può essere trovato in una locazione di memoria identificata dalla label CNT. Questo buffer sorgente è equivalente al buffer sorgente nel programma di movimento dati appena descritto.

La destinazione del blocco di dati è una delle tabelle dati. Il numero di tabella è identificato dal simbolo TBN0 che è caricato come dato immediato. I primo due bytes di ogni tabella identificano lo spostamento del primo byte libero della tabella; in altre parole, si assumerà che ogni tabella sia parzialmente riempita ed il blocco di dati sia mosso nella parte finale della tabella selezionata.
Il movimento dati richiesto può essere illustrato come segue:



Ecco una sequenza d'istruzione appropriata:

```
      LHLD  TABX+TBN0*2-2
      MOV   E,M       ; Carica lo spostamento del primo byte libero in DE
      INX   H
      MOV   D,M
      DAD   D         ; Somma ad HL, originando l'indirizzo del primo byte libero
      LXI   D,CBASE   ; Carica la base del buffer d'ingresso in DE
      LDA   CNT       ; Carica il contatore di byte e conserva in B
      MOV   B,A
LOOP  LDAX  D         ; Muove il byte successivo dalla locazione di memoria
```

```
MOV     M,A     ; Indirizzata da DE alla locazione indirizzata da HL
INX     D       ; Incrementa la sorgente e gli
INX     H       ; Indirizzi di destinazione
DCR     B       ; Decrementa il contatore di byte
JNZ     LOOP    ; Ritorno per più bytes
```

CLASSIFICAZIONE DI DATI

Entrambi gli esempi di programmazione descritti muovono semplicemente un blocco di dati da una locazione ad un'altra. Anche la riorganizzazione di dati è molto importante perciò **si illustrerà una routine di classificazione.**

La classificazione così illustrata, considera una sequenza di numeri binari con segno, immagazzinati in locazioni di memoria contigue, li riorganizza in ordine crescente, cosicchè il numero più piccolo diviene il primo ed il numero più grande l'ultimo.

La routine di classificazione che si vuole programmare impiega un algoritmo bubble-up. Si consideri una sequenza di numeri dove la label LIST identifica l'indirizzo della locazione di immagazzinamento del primo numero in memoria. Le fasi del programma della routine di classificazione sono le seguenti:

SCELTA DEI DATI

1) Inizia un passo all'inizio della LIST, inizializza un flag per indicare una condizione "non scambio".
2) Confronta una coppia successiva di numeri; se il primo numero è più piccolo del secondo, non opera altrimenti scambia i due numeri e pone il flag per indicare "eseguito scambio".
3) Confronta l'indirizzo del secondo numero all'indirizzo della fine della lista, identificato mediante la label ENDL. Se non è la fine incrementa cosicchè il secondo numero della coppia attuale diventa il primo numero della coppia successiva e ritorna alla fase 2.
4) Alla fine della lista controlla il flag di "scambio". Se qualsiasi scambio è stato fatto durante il passo ritorna alla fase 1 per fare un altro passo.
5) Se è stato fatto un passo senza scambi tutti i numeri sono in ordine. Uscita.

Come esempio si consideri il caso in cui i numeri da 1 a 10 siano in ordine inverso. Durante il primo passo saranno fatti nove scambi, alla fine del quale il numero più grande sarà stato inviato alla sommità:

	INIZIO	DOPO 1 PASSO
LIST	10	9
	9	8
	8	7
	7	6
	6	5
	5	4
	4	3
	3	2
	2	1
ENDL	1	10

Altri otto passi saranno richiesti per porre tutti i numeri in ordine, quindi il decimo passo è richiesto per acquisire una condizione di uscita di "non scambio".

SORT è realizzata come una subroutine che ha i parametri di passaggio nelle locazioni seguenti la chiamata di subroutine. Sono specificati due parametri.

LIST	L'indirizzo iniziale del buffer dati contenente i numeri da classificare
ENDL	l'indirizzo finale del buffer dati contenente i numeri da classificare

Ecco il programma di classificazione:

```
        CALL  SORT   ; Chiama la sequenza
        DW    LIST   ; Indirizzo dell'inizio della lista
        DW    ENDL   ; Indirizzo della fine della lista (sulla stessa pagina come lista)
        –
        –
        –
SORT    POP   H      ; Indirizzo di ritorno su Stack in HL
        MOV   E,M    ; Carica l'indirizzo della lista DE
        INX   H
        MOV   D,M
        INX   H
        MOV   D,M    ; Carica il byte di basso ordine di indirizzo ENDL a C
        INX   H
        INX   H      ; Incrementa il byte di alto ordine passato di indirizzo ENDL
        PUSH  H      ; Ritorno Stack
        MOV   H,D    ; Muove il byte di alto ordine della lista da D ad H
        MVI   D'0    ; Lo zero di D è l'indicatore di "non scambio"
LOOP1   MOV   L,E    ; Muove il byte di basso ordine dell'indirizzo della lista da
                     ; E ad L
LOOP2   MOV   A,M    ; Carica in A il primo numero della coppia
        INR   L      ; Incrementa il puntatore della lista
        CMP   M      ; Confronta (sottrae la memoria da A)
        JM    SORT1  ; Salta se meno (secondo numero già maggiore del primo)
        MOV   B,A    ; Muove il primo numero da B ad A
        MOV   A,M    ; Carica in A il secondo numero
        DCR   L      ; Decrementa alla locazione del primo numero
        MOV   M,A    ; Immagazzina il secondo numero da A
        INR   L      ; Incrementa il puntatore della lista
        MOV   M,B    ; Immagazzina il primo numero da B
        MVI   D,1    ; Carica D con la costante 1 come un flag di "scambio ese-
                     ; guito"
SORT1   MOV   A,C    ; Muove il byte di basso ordine dell'indirizzo ENDL ad A
        CMP   L      ; Confronta col byte dell'indirizzo della lista in L
        JNZ   LOOP2  ; Ricicla indietro se non è il byte finale
        DCR   D      ; Decrementa il flag in D
        JZ    LOOP1  ; Ricicla indietro all'inizio della lista se è fatto uno scambio
        RET          ; Ritorno
```

ARITMETICA

In questo paragrafo saranno descritte l'addizione, sottrazione, moltiplicazione e divisione. Le funsioni trascendenti sono abbastanza complesse da richiedere l'intero testo di un libro rivolto all'argomento, cosicchè si introdurrà soltanto su questo argomento.

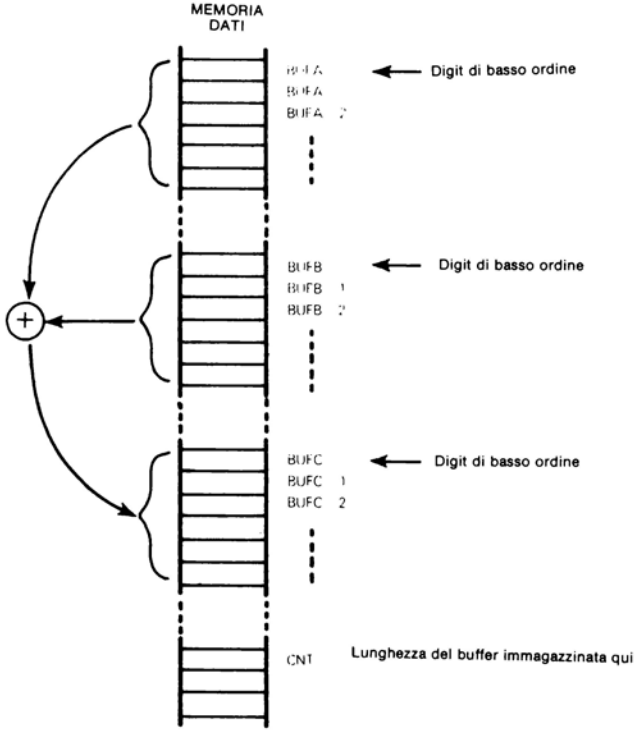
Anche all'interno dei semplici limiti di addizione, sottrazione, moltiplicazione e divisione, c'è un grado di latitudine che eccede lo scopo di materiale che si può sviluppare. Algoritmi significativamente diversi sono richiesti in dipendenza della grandezza del numero. Inoltre l'aritmetica binaria e decimale richiedono algoritmi diversi. Perciò **per addizione e sottrazione, si considereranno numeri grandi o piccoli, binari o decimali. Per moltiplicazioni e divisioni si considereranno soltanto numeri binari piccoli.**

ADDIZIONE BINARIA

Si considera prima l'addizione binaria multibyte.

Due numeri positivi interi, ciascuno lungo CNT bytes, devono essere sommati. Gli indirizzi di partenza del buffer dal numero sono forniti da BUF1 e BUF2. Il risultato deve essere immagazzinato in un buffer iniziante in BUF3.

L'addizione multibyte può essere illustrata come segue:

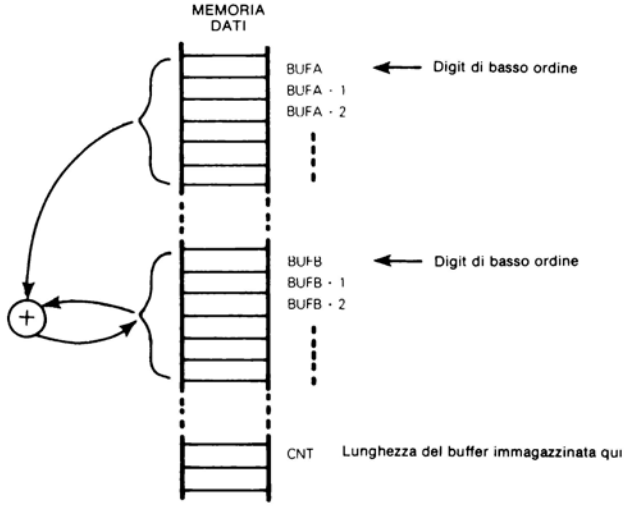


Questa sequenza di istruzioni esegue l'addizione illustrata:

```
        LDA   CNT    ; Carica la lunghezza di buffer e conserva in C
        MOV   C,A
        LXI   H,BUFC ; Carica l'indirizzo di buffer del risultato in H ed L
        PUSH  H      ; Consrva sullo Stack
        LXI   D,BUFA ; Carica l'indirizzo del primo buffer in D ed E
        LXI   H,BUFB ; Carica l'indirizzo del secondo buffer in H ed L
        XRA   A      ; Azzera il Carry
LOOP    LDAX  D      ; Carica il byte successivo di BUF1
        ADC   M      ; Somma il byte successivo di BUF2
        XTHL         ; Conserva nel byte successivo del buffer del risultato
        MOV   M,A
        INX   H      ; Incrementa l'indirizzo di BUFC
        XTHL
        INX   D      ; Incrementa l'indirizzo di BUF1
```

```
        INX     H       ; Incrementa l'indirizzo di BUFB
        DCR     C       ; Decrementa il contatore
        JNZ     LOOP    ; Ritorno per più bytes
```

L'addizione multibyte è più semplice se si può immagazzinare la somma in uno dei buffers sorgente:



Ecco la sequenza d'istruzione più breve:

```
        LDA     CNT     ; Carica la lunghezza di buffer e conserva in C
        MOV     C,A
        LXI     D,BUFA  ; Carica l'indirizzo del primo buffer in D, E
        LXI     H,BUFB  ; Carica in H, L gli indirizzi del buffer del risultato ed il
                        , secondo
        XRA     A       ; Azzera Carry
LOOP    LDAX    D       ; Carica il byte BUFA successivo
        ADC     M       ; Somma il byte BUFB successivo
        MOV     M,A     ; Immagazzina la risposta
        INX     D       ; Incrementa l'indirizzo BUFA
        INX     H       ; Incrementa l'indirizzo BUFB
        DCR     C       ; Decrementa la lunghezza del buffer
        JNZ     LOOP    ; Ritorno se non è finito
```

SOTTRAZIONE BINARIA

Poichè l'8080 ha istruzioni speciali per la sottrazione, la sottrazione binaria è pressochè identica all'addizione binaria. Nella stessa subroutine si sostituisce semplicemente l'istruzione ADC con SBB e risulterà la sottrazione binaria richiesta.

ADDIZIONE DECIMALE

Anche l'addizione decimale è molto facile impiegando un microcalcolatore 8080. **Si inserisce semplicemente un'istruzione DAA a seguito della ADC** nei programmi dall'addizione binaria e si avrà l'addizione decimale:

```
        –
        –
        –
LOOP    LDAX  D      ; Carica il byte successivo di BUF1
        ADC   M      ; Somma il byte successivo di BUF2
        DAA          ; Aggiusta il risultato decimale
        XTHL         ; Conserva nel byte successivo del buffer del risultato
        –
        –
        –
```

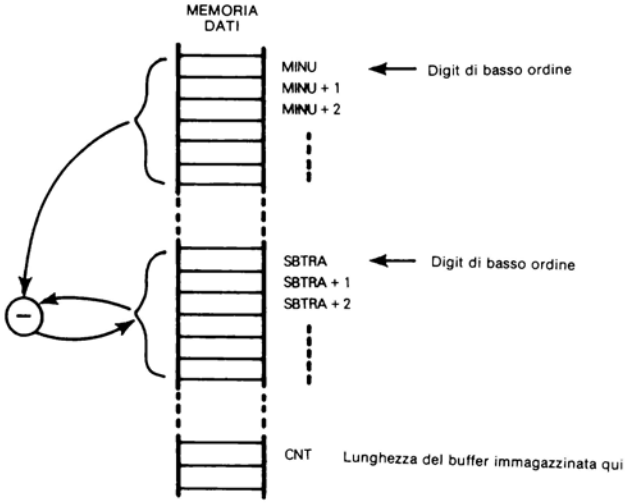
Comunque è necessario una cautela: la routine dell'addizione decimale costruita assume che i dati decimali validi codificati binari siano immagazzinati nei buffers sorgente. Se, per errore, si ha un dato non valido in uno dei buffers sorgente si genererà un risultato privo di significato – e non si conoscerà.

Se il programma è tale da non poter garantire che i dati nei buffers sorgente siano decimali codificati binari validi, allora si deve scrivere una routine per controllare i contenuti dei buffers ed assicurare che nessuna unità alta o bassa di 4-bit all'interno di qualsiasi byte contenga un codice binario da A ad F.

SOTTRAZIONE DECIMALE

La sottrazione decimale è alquanto complicata dal fatto che non si possono impiegare le istruzioni di sottrazione dell'8080; queste istruzioni lavorano solo per dati binari poichè esse generano automaticamente il complemento a due del sottraendo. Come descritto in "An Introduction to Microcomputer", Volume I la sottrazione in decimale codificato binario richiede la considerazione del complemento a dieci del sottraendo.

Si ritorni al programma dell'addizione binaria e si costruisca, al suo posto, un equivalente sottrazione decimale; ecco la mappa di memoria opportuna:



Ecco la sequenza d'istruzione richiesta:

```
DSUB: LXI  D,MINU  ; D ed E indirizzo minuendo
      LXI  H,SBTRA ; H ed L indirizzo sottranedo
      LDA  CNT     ; Carica la lunghezza del buffer e la conserva
      MOV  C,A     ; In C
      STC          ; Pone il Carry indicante non prestito
LOOP: MVI  A,99H   ; Carica l'Accumulatore con 99H
      ACI  0       ; Somma zero con Carry
      SUB  M       ; Produce complemento a nove del sottraendo
      XCHG         ; Commuta D ed E con H ed L
      ADD  M       ; Somma il minuendo
      DAA          ; Aggiusta i decimali dell'Accumulatore
      XCHG         ; Commuta ancora D ed E con H ed L
      MOV  M,A     ; Immagazzina il risultato
      INX  D       ; Indirizzo del byte successivo del minuendo
      INX  H       ; Indirizzo del byte successivo del sottraendo
      DCR  C       ; Decrementa il byte del conteggio
      JNZ  LOOP    ; Accetta i successivi 2 digits decimali
DONE: NOP
```

MOLTIPLICAZIONE E DIVISIONE

Per la moltiplicazione e divisione occorre una particolare cautela nell'eseguire all'interno di un sistema a microcalcolatore. Queste sono operazioni non adatte all'organizzazione di un microcalcolatore; qualsiasi moltiplicazione o divisione non banale può essere così lunga da eseguire che essa degraderà notevolmente l'esecuzione globale. **Se l'applicazione del microcalcolatore che si considera fa uso estensivo di moltiplicazioni, divisioni o funzioni trascendenti si dovrebbe considerare seriamente l'impiego di uno dei molti chips di calcolo aritmetico che ora sono commercialmente disponibili.** Trasferendo l'aritmetica complessa a tale chip si può fare differenza tra un sistema a microcalcolatore impiegabile oppure no nell'applicazione che si considera.

Si può realizzare semplicemente la moltiplicazione e divisione nei sistemi a microcalcolatore che non fanno un impiego estensivo, o senza problemi di consumo di tempo, di queste routine; perciò si descriverà qualche sequenza semplice di programma.

MOLTIPLICAZIONE BINARIA AD 8-BIT

Si consideri la moltiplicazione di due valori dati ad 8-bit, senza segno, per generare un prodotto a 16-bit. Il modo più semplice per eseguire questa moltiplicazione è di sommare il moltiplicare a 0 il numero di volte dato dal moltiplicando. Per esempio, **si può moltiplicare 4 per 3 se si somma 4 a 0 tre volte:**

```
      MVI  A,0     ; Azzera AB per inizializzare il buffer
      MOV  B,A     ; Del risultato
      CMP  D       ; Prova per 0 in D
      RZ           ; Se 0, il risultato è 0 e così finisce
LOOP  ADD  E       ; Somma il moltiplicare al byte del risultato di basso ordine
      JNX  NEXT    ; Se il Carry è 1, incrementa B
      INR  B
NEXT  DCR  D       ; Decrementa il moltiplicando
      JNZ  LOOP    ; Se non è zero, ritorno per sommare ancora
      RET
```

Esiste un modo più veloce di esecuzione delle moltiplicazioni. Si può impiegare il fatto che un digit binario è limitato ad essere 0 od 1; questo significa che al livello di semplice digit, la moltiplicazione degenera all'addizione o non addizione.

Si spiega questo concetto; impiegando la comune notazione decimale, si consideri la seguente moltiplicazione:

```
      142        Moltiplicando
    x 307        Moltiplicatore
   -------
    42600
     0000       Prodotto Parziale
      994
   -------
    43594       Prodotto
```

142 = Moltiplicatore
307 = Moltiplicando

Somma al prodotto 7 x Moltiplicatore

Sposta il Moltiplicatore di due digit a sinistra, quindi moltiplica x 3 e somma al prodotto

Ogni prodotto parziale è uguale alla moltiplicazione del moltiplicando mediante un digit del moltiplicatore. Il prodotto parziale, è spostato a sinistra aggiungendo zeri a destra. Il numero di zeri a destra è uguale al numero di digits alla destra dell'attuale digit moltiplicatore:

```
  142
  3xx
42600
```

Aggiungi due zeri perchè ci sono due digits alla destra di 3

142 x 3

Si può estendere lo stesso concetto all'aritmetica binaria nel cui caso il problema diviene molto semplice, poichè nessun digit binario può avere valori diversi da 0 a 1. In questo caso si hanno solo due scelte: dovunque un digit moltiplicatore è 0 non si somma il moltiplicando spostato al risultato; ma se il digit moltiplicatore è 1 si somma il moltiplicando spostato al risultato. Ecco un esempio:

```
10110101 = Moltiplicando (M)
01101101 = Moltiplicatore
```

Somma M al prodotto

Sposta M di due digit a sinistra e somma

Sposta M di tre digits a sinistra e somma

Sposta M di cinque digits a sinistra e somma

Sposta M di sei digits a sinistra e somma

```
Prodotto =        10110101
                 101101010
                 -----------
                 1110001001
                1011010100
               -------------
               100100110001
              1011010100000
             ---------------
             111111110001
            10110101000000
           -----------------
           100110100010001
           4   D   1   1
```

```
10110101 x 01101101 = 0100110100010001
B   5      6   D      4   D   1   1
```

Impiegando la tecnica "sposta-e-somma", le fasi seguenti moltiplicheranno un moltiplicando ad un byte con un moltiplicatore ad un byte per produrre il risultato corretto a due bytes:

a) Prova il bit meno significativo del moltiplicatore. Se zero, va alla Fase b. Se uno somma il moltiplicando al byte più significativo del risultato.
b) Sposta l'intero risultato a due bytes a destra di una posizione di bit.
c) Ripete le Fasi a e b finchè tutti i bit del moltiplicatore non sono stati provati.

Si consideri B5 * D6, la moltiplicazione binaria appena illustrata:

Moltiplicatore = 01101101

Moltiplicando = 10110101

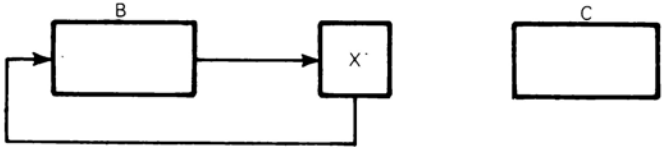
			RISULTATO: BYTE DI ORDINE ELEVATO	RISULTATO: BYTE DI BASSO ORDINE
	Inizio		0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 1	Fase 1 (a)		1 0 1 1 0 1 0 1	0 0 0 0 0 0 0 0
	1 (b)		0 1 0 1 1 0 1 0	1 0 0 0 0 0 0 0
0 1 1 0 1 1 0 1	Fase 2 (a,b)		0 0 1 0 1 1 0 1	0 1 0 0 0 0 0 0
0 1 1 0 1 1 0 1	Fase 3 (a)		1 0 1 1 0 1 0 1	
			1 1 1 0 0 0 1 0	0 1 0 0 0 0 0 0
	3 (b)		0 1 1 1 0 0 0 1	0 0 1 0 0 0 0 0
0 1 1 0 1 1 0 1	Fase 4 (a)		1 0 1 1 0 1 0 1	
		c→1	0 0 1 0 0 1 1 0	0 0 1 0 0 0 0 0
	4 (b)		1 0 0 1 0 0 1 1	0 0 0 1 0 0 0 0
0 1 1 0 1 1 0 1	Fase 5 (a,b)		0 1 0 0 1 0 0 1	1 0 0 0 1 0 0 0
0 1 1 0 1 1 0 1	Fase 6 (a)		1 0 1 1 0 1 0 1	
			1 1 1 1 1 1 1 0	1 0 0 0 1 0 0 0
	6 (b)		0 1 1 1 1 1 1 1	0 1 0 0 0 1 0 0
0 1 1 0 1 1 0 1	Fase 7 (a)		1 0 1 1 0 1 0 1	
		c→1	0 0 1 1 0 1 0 0	0 1 0 0 0 1 0 0
	7 (b)		1 0 0 1 1 0 1 0	0 0 1 0 0 0 1 0
0 1 1 0 1 1 0 1	Fase 8 (a,b)		0 1 0 0 1 1 0 1	0 0 0 1 0 0 0 1
			4 D	1 1

Si scriverà ora un programma per realizzare questo algoritmo di moltiplicazione.

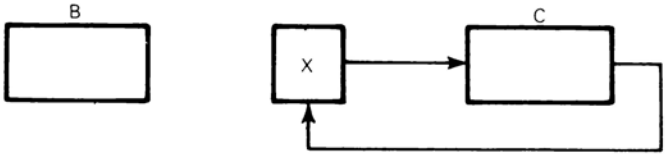
Il Registro B conserverà il byte più significativo del risultato, ed il registro C conterrà il byte meno significativo del risultato.

Lo spostamento a 16-bit a destra del risultato è eseguito mediante due istruzioni ruota-a-destra-attraverso-carry come segue:

Ruota B:



Quindi ruota C per completare lo spostamento:



Il registro D conserva il moltiplicando ed il registro C originariamente conserva il moltiplicatore. Ecco il programma:

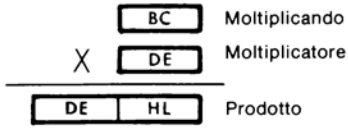
```
MULT  MVI   B,0     ; Inizializza il byte più significativo del risultato
      MVI   E,9     ; Contatore di bit
MULT0 MOV   A,C     ; Ruota il bit meno significativo del
      RAR           ; Moltiplicatore al Carry e sposta
      MOV   C,A     ; Il byte di basso ordine del risultato
      DCR   E
      JZ    DONE    ; Esce se completo
      MOV   A,B
      JNC   MULT1
      ADD   D       ; Somma il moltiplicando al byte di basso ordine del risul-
                    ; tato se il bit era uno
MULT1 RAR           ; Carry=0 qui; sposta il byte di ordine elevato del risultato
      MOV   B,A
      JMP   MULT0
DONE
```

MOLTIPLICAZIONE BINARIA A 16-BIT

Si consideri ora la moltiplicazione di due numeri a 16-bit, che produce un risultato a 32-bit.

Questo è l'algoritmo impiegato:

1) Sposta il moltiplicatore (Alto Ordine 16 bit) ed il prodotto parziale (Basso Ordine 16 bit) a sinistra attraverso il carry.
2) Somma il moltiplicando a 16-bit ai tre bytes del prodotto parziale se un bit uno è spostato fuori del moltiplicatore nel carry.

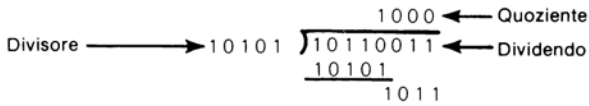


Ecco le istruzioni necessarie:

```
MPY   LXI   H,0     ; Inizializza a zero il prodotto parziale in HL
      MVI   A,16    ; Inizializza il conteggio
LOOP  DAD   H       ; Somma HL ad HL-spostamento logico a sinistra nel carry
      XCHG          ; Scambia HL e DE
      JC    MPY1    ; Salta se carry è fuori da HL
      DAD   H       ; No carry-spostamento logico a sinistra del moltiplicatore
                    ; nel carry
      JMP   MPY2    ; Salta
MPY1  DAD   H       ; Carry-spostamento logico a sinistra del moltiplicatore nel
                    ; carry
      INX   H       ; Ed incremento
MPY2  XCHG          ; Ripunta al prodotto parziale
      JNC   MPY3    ; Salta se non somma (bit del moltiplicatore nel carry=0)
      DAD   B       ; Somma il moltiplicando in BC al prodotto parziale in HL
      JNC   MPY3    ; Salta se il carry non è fuori
      INX   D       ; Incrementa DE per sommare carry
MPY3  DCR   A       ; Decrementa il conteggio
      JNZ   LOOP    ; Ricicla se non zero
      RET           ; Ritorno
```

DIVISIONE BINARIA

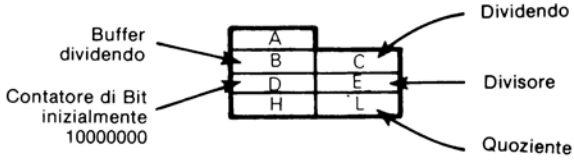
Si consideri la semplice divisione ad 8-bit. $B3_{16}$ diviso da 15_{16} può essere illustrato come segue:



Il risultato è 8_{16} con un resto di B_{16}.

L'algoritmo della divisione opera spostando a sinistra il dividendo nel registro che è inizialmente azzerato. Ogni volta che i contenuti del buffer di spostamento del dividendo eccedono il divisore, viene sottratto il divisore dai contenuti del buffer di spostamento ed un digit binario 1 è inserito nella poisizione di bit appropriata al quoziente.

Si consideri la seguente assegnazione di registri:



Si assuma inizialmente che il divisore è nel registro E ed il dividendo nel Registro C. Il quoziente sarà generato nel Registro L. Ecco il programma della divisione che risulta:

```
; Inizialmente azzera i registri A, B, L e Carry
        XRA     A       ; OR esclusivo di A con sé stesso. Questo azzera A
        MOV     B,A     ; Azzera B
        MOV     L,A     ; Azzera L
; Inizializza il contatore di bit nel registro 0 D
        MVI     D0,80H
; Sposta B e C, come unità a 16-bit, di un bit a sinistra
LOOP    MOV     A,C
        RAL
        MOV     C,A
        MOV     A,B
        RAL
        MOV     B,A
; Confronta il divisore (in E) con il dividendo, sposta il buffer,
; Attualmente fermo
        CMP     E
        JC      NEXT    ; Se il divisore è maggiore, scavalca la sottrazione
; Il divisore è più piccolo, sottrae dal
; Dividendo il buffer spostato
        SUB     E
        MOV     B,A
, Pone ad 1 il bit attuale di L. Il bit attuale è
; Il bit di posizione 1 in D
        MOV     A,D
        ORA     L
        MOV     L,A
; Sposta D a destra di una posizione di bit ed azzera Carry
NEXT    MOV     A,D
        RRC
        MOV     D,A
        JNC     LOOP    ; Se Carry non è 1, ritorno per il bit successivo
```

Alla fine il quoziente sarà in L mentre il resto sarà in B.

LOGICA DELLA SEQUENZA DI ESECUZIONE DEL PROGRAMMA

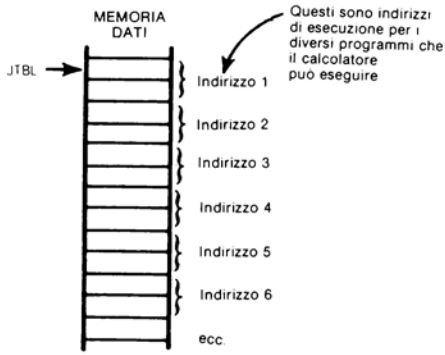
LA TABELLA DI SALTO

C'è in realtà solo una sequenza di programma che richiede di essere descritta in questo contesto; è la Tabella di Salto.

Si ricordi che il set di istruzione dell'8080 è ricco di istruzioni condizionali: le istruzioni di Salto, Chiamata e Ritorno hanno in tutto otto variazioni condizionali, questo significa che non sono richieste speciali routine quando la logica necessaria può avere solo uno o due modi.

Quando si hanno tre o più scelte, la Tabella di Salto diviene uno strumento di programmazione reale.

Nel cuore di una Tabella di Salto ci sarà una sequenza di indirizzi a 16-bit immagazzinati in coppie di bytes di memoria contigui:



Si presumerà che questi indirizzi di memoria contigui rappresentino l'indirizzo di partenza per un certo numero di programmi diversi. Assumendo che il programma richiesto è identificato da un numero di programma contenuto nell'Accumulatore, **la seguente sequenza di istruzione origina l'esecuzione di diramazione al programma il cui numero è immagazzinato nell'Accumulatore:**

Programma della tabella di salto

```
LXI   H,JTBL  ; Carica l'indirizzo base della tabella di salto in HL
ADD   A       ; Moltiplica l'Accumulatore per 2
ADD   L       ; Somma ad A il byte di indirizzo di basso ordine
MOV   L,A     ; Ri-immagazzina SUM in L
MVI   A,0     ; Somma Carry (se qualunque) ad H
ADC   H
MOV   H,A
MOV   E,M     ; HL indirizza l'indirizzo richiesto
INX   H       ; Carica l'indirizzo in D, E
MOV   D,M
XCHG          ; Muove l'indirizzo da DE ad HL
PCHL          ; Muove l'indirizzo da HL a PC
```

Appendice A

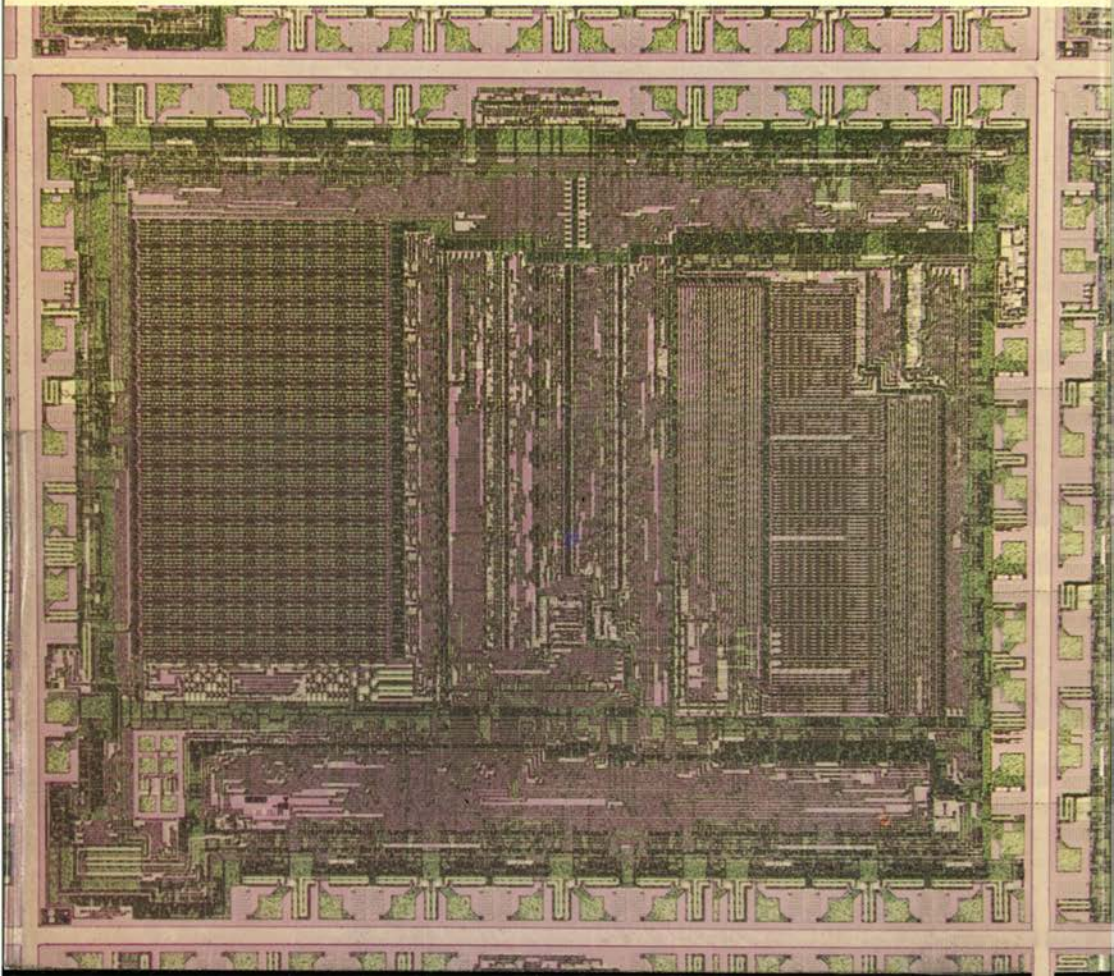
CODICI DI CARATTERE ASCII

b7 →					0	0	0	0	1	1	1	1
b6 →					0	0	1	1	0	0	1	1
b5 →					0	1	0	1	0	1	0	1
BITS	b4	b3	b2	b1	Colonna / Riga	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P		p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FS	,	<	L	\	l	¦
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

NUL	Nullo
SOH	Inizio della testata
STX	Inizio del testo
ETX	Fine del testo
EOT	Fine della trasmissione
ENQ	Indagine
ACK	Riconoscimento
BEL	Campanello o allarme
HT	Tabulazione orizzontale
LF	Alimentazione linea
VT	Tabulazione verticale
FF	Alimentazione form
CR	Ritorno carrello
SO	Sposta fuori
SI	Sposta dentro
DLE	Dati uscita dal link
DC1	Controllo dispositivo 1
DC2	Controllo dispositivo 2
DC3	Controllo dispositivo 3
DC4	Controllo dispositivo 4
NAK	Riconoscimento negativo
STN	Attesa sincrona
ETB	Fine del blocco di trasmissione
CAN	Cancello
EM	Fine del mezzo
SUB	Sostituto
ESC	Fuga
FS	Separatore di fila
GS	Separatore di gruppo
RS	Separatore di disco
US	Separatore di unità
SP	Spazio
DEL	Cancella

L. 16.500

Cod. 325 P



35

Programmazione dell'8080 e progettazione logica

Adam OSBORNE

GRUPPO EDITORIALE JACKSON